eoe•Android开发者社区、ChinaUnix技术社区鼎力推荐
讲解生动，内容浅显易懂，零门槛学Android编程，真的很简单



清华大学出版社



清华大学出版社
北 京

内 容 简 介

本书是一本与众不同的 Android 学习读物，是一本化繁为简，把抽象问题具体化，把复杂问题简单化的书。本书避免出现云山雾罩、晦涩难懂的讲解，代之以轻松活泼、由浅入深的剖析。这必将使得阅读本书的读者少走弯路，快速上手，从而建立学习 Android 开发的信心。本书配带 1 张光盘，收录了本书重点内容的教学视频和本书涉及的所有源代码。

本书共 14 章，分为 4 篇。主要内容涵盖了 Android 发展现状、开发环境的搭建、开发工具的使用、Android 工程结构的剖析、UI 界面的设计方法及各个常用功能的实现，最后介绍了两个综合项目案例的开发过程。通过阅读本书，读者可以在较短的时间内理解 Android 开发的各个重要概念和知识点，为进一步学习打好基础。

本书适合没有接触过 Android 开发的新手阅读，但建议读者阅读本书前对 Java 编程有一定的了解；对于有一定经验的 Android 开发人员，也可以通过本书进一步理解 Android 语言的各个重要知识点和概念。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

Android 编程入门很简单 / 王勇等编著. —北京：清华大学出版社，2012.7
（入门很简单丛书）
ISBN 978-7-302-28866-4

Ⅰ. ①A… Ⅱ. ①王… Ⅲ. ①移动终端—应用程序—程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2012）第 104730 号

责任编辑：夏兆彦
封面设计：欧振旭
责任校对：徐俊伟
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**27.75　　**字　　数：**690 千字
附光盘 1 张
版　　次：2012 年 8 月第 1 版　　**印　　次：**2012 年 8 月第 1 次印刷
印　　数：1～
定　　价：　　元

产品编号：047217-01

前　　言

在开放手机联盟（Open Handset Alliance，OHA）的大力推动下，一个时尚、热门、免费并开源的移动平台——Android 正在飞速发展。越来越多的厂商开始关注 Android，越来越多的用户选择使用 Android。与此同时，越来越多的开发者正在投入到 Android 开发大军。在这样的背景下，本书应运而生。它可以帮助那些对 Android 开发有兴趣的人快速进入 Android 移动开发领域。如果您已经是一个资深的移动应用开发者，本书也可以帮助你再次梳理 Android 开发中需要掌握的一些知识点。

为何选择 Android 开发平台

如今，市场上已经有了许多移动开发平台，包括 Symbian、iPhone、Windows Mobile、BlackBerry、Java Mobile Edition 和 Linux Mobile（LiMo）等。当笔者向别人说起 Android 时，他们的第一个疑问通常是：我们为什么还需要另一个移动标准？它有何惊人之处？

虽然 Android 的一些特性并非首创，但它是第一个将以下特性结合在一起的环境。

1. 基于Linux，真正开放、开源、免费的开发平台

手持设备制造商钟情于 Android 的原因，是它们可以使用和定制该平台而不需要支付费用。开发人员喜欢 Android 的原因，是他们知道该平台是独立的，不受任何厂商的限制。

2. 受Internet mashup思想启发的基于组件的架构

基于 Android 开发平台，一个应用程序的组件可以在另一个应用程序中用作其他用途，甚至可以将 Android 内置的组件替换为自己改进后的版本。这将在移动领域掀起新一轮的创造风潮。

3. 众多开箱即用的内置服务

Android 基于位置的服务使用 GPS 或手机发射塔三角测量法，让你可根据所处位置来定制用户体验；凭借功能全面的 SQL 数据库，利用强大的本地存储，可以完成偶尔连接的计算和同步操作；浏览器和地图视图可以直接嵌入到应用程序中。所有这些内置服务有助于提高功能的标准，同时降低开发成本。

4. 应用程序生命周期的自动化管理

Android 的多层安全措施将程序彼此分离，这将使智能电话的系统稳定性达到前所未有的水平。最终用户不再需要担心哪些应用程序是活动的，也不必在运行新程序前关闭原

有的一些程序。Android 针对低能耗、低内存的设备进行了优化，这种根本性的优化是之前的平台从未尝试过的。

5．高质量的图形和声音

Android 将类似于 Flash 的光滑、无锯齿的 2D 矢量图形和动画与 3D 加速的 OpenGL 图形相结合，可实现各种新式的游戏和商业应用程序。Android 内置了最常用的行业标准音频和视频格式的编解码器，这些格式包括 H.264（AVC）、MP3 和 AAC。

6．当前及未来各类硬件间的可移植性

Android 平台的所有程序都是用 Java 语言编写的，并且由 Android 的 Dalvik 虚拟机执行，所以其代码在 ARM、X86 和其他架构之间是可以移植的。Android 提供了对各种输入法的支持，如键盘、触摸屏和轨迹球等。用户界面可以针对任何屏幕的分辨率和屏幕方向进行定制。

本书写作的目的，是通过对 Android 程序设计基础知识和基本技能系统而全面的讲解，使读者能够轻松掌握 Android 程序设计的基本知识和技能，尽量减少在 Android 程序设计入门阶段的摸索和徘徊，为进一步学习 Android 程序设计高级技术打下坚实的基础。

本书有何特色

1．提供配套的多媒体教学视频

本书中的重点内容都录制了配套的多媒体教学视频，以帮助读者更加直观而高效地学习，从而达到事半功倍的效果。

2．讲解通俗易懂，入门非常容易

本书不介绍初学者不需要的技术和操作，也不会云山雾罩地分析问题。笔者坚信首先应该细嚼慢咽地掌握基本原理，理解基本概念，然后才能更进一步学习。一旦打好了基础，“更难”的部分看起来也就没那么难了。本书将会让读者真正地轻松入门。

3．内容全面，穿插大量实例，讲解方法丰富

本书对基础概念都做了全面而详细的解析，并对重要概念和比较难理解的知识提供了实际的例子进行讲解。其中用到了类比、比喻等讲解方法，并且给出了形象的图示，以加深读者的理解。

4．图解教学

对于 Android 开发中一些比较难于理解的内容，本书采用多插图的形式，用更加形象、风趣和直观的方式讲解，利于初学者的学习和理解。

5．风格清新，趣味讲解，提高易读性

已经出版的 Android 编程图书，大多板着个面孔，平淡无趣，拒读者于千里之外。本

书试图用清新活泼的风格，并适当结合幽默的语言，来激发读者的阅读兴趣。

6．举一反三

本书不是知识点的简单罗列，而是让读者学会一个知识点后编写相应的代码，并且进行拓展，应用到相同类型的开发中，做到举一反三、授人以渔的效果。

7．配合项目案例教学，提高实战开发水平

本书尽力消除了初学者学习计算机语言时所能遇到的障碍，变抽象为具体，变复杂为简单。这是一本入门书，如果你还从来没有写过 Android 程序，那么这本书正好适合你。

本书内容概览

第1篇　入门必备（第1~4章）

本篇简单讲述了 Android 开发现状、本书的学习曲线、开发环境的安装及各类开发工具的使用，并尝试新建了第一个 Android 工程。通过学习本篇内容，读者可以对本书的学习方法有一个初步的了解，并对 Android 编程有一个宏观的认识。

第2篇　界面开发（第5~7章）

本篇主要讲述了 Android 开发中的界面开发部分，主要分为以下 3 个方面：

（1）各种视图的类的使用；

（2）各类资源的调用；

（3）Android 的 5 类布局的合理嵌套。

读者在学习完本篇后可以熟练地进行程序界面的设计和实现。

第3篇　功能实现（第8~12章）

本篇讲述了 Android 开发中一些比较复杂的技术，也可以称之为高级技术，这些看似稍微复杂的技术也正是 Android 开发的核心。能否使用 Android SDK 游刃有余地进行开发，就要看对本篇内容的理解和掌握程度了。

第4篇　项目案例开发（第13、14章）

本篇主要通过两个实际的项目案例，帮助读者将本书前面所学的知识点进行系统的应用。通过本篇的实战开发，读者就可以进行实际的 Android 开发了。

本书为谁而写

本书最为适合 Android 编程入门人员阅读，但建议读者阅读本书前有一定的 Java 编程基础。本书的读者主要有以下几类：

- ❑ Android 开发初学者；

- Android 移动开发从业人员；
- 大中专院校的学生；
- 相关培训班的学员；
- Android 开发爱好者。

本书作者

本书由王勇主笔编写，其他参与编写的人员有陈世琼、陈欣、陈智敏、董加强、范礼、郭秋滟、郝红英、蒋春蕾、黎华、刘建准、刘霄、刘亚军、刘仲义、柳刚、罗永峰、马奎林、马味、欧阳昉、蒲军、齐凤莲、王海涛、魏来科、伍生全等。

您在阅读本书的过程中若有疑问，请发 E-mail 和我们联系。E-mail 地址：bookservice2008@163.com。

目　　录

第 1 篇　入 门 必 备

第 2 篇 界 面 开 发

第 3 篇 功 能 实 现

第 4 篇 项目案例开发

第 1 篇　入门必备

⏭　第 1 章　初识 Android

⏭　第 2 章　搭建你的开发环境

⏭　第 3 章　创建第一个程序——HelloWorld

⏭　第 4 章　使用 Android 工具

第 1 章　初识 Android

如今的 Android 毫不夸张地说已经红透了半边天。越来越多的 Android 用户使得 Android 的市场需求与日俱增：用户们希望得到更强大的功能、更丰富的应用、更个性化的手机，这无疑是 Android 手机开发者的福音！它是一个稳定、安全而又开放的平台，同时也是一个充满了机遇和挑战的舞台。本章我们将了解到：

（1）什么是 Android。

（2）Android 的历史以及其发展趋势。

（3）为什么 Android 能这么火。

1.1　手机发展简史

现在人们的生活已经越来越无法离开手机。设想一下，你无论是在上课、上班或者逛街，你的身边是不是一定有一部手机在陪伴着你？当你无聊的时候你会拿出手机玩游戏，又或者找好友聊天，甚至你可以随时上网听音乐看电影！是啊，现在的手机功能如此强大，可是手机从一开始就有这些功能吗？

1.1.1　手机发展的里程碑

让我们重新回顾手机的发展历程，这里仅以那些重要的里程碑式的事件作为线索，带领读者朋友们回到那些激动人心的年代：

1．1875年6月2日：第一部电话诞生

1875 年 6 月 2 日，经过了一段时间的研究和努力，贝尔和沃森终于完成了他们的电话模型。贝尔在一间房子里做最后的准备，而沃森则在另一间屋子里关着门窗，耳朵紧贴音箱准备接听。这时，贝尔不小心将硫酸洒到了大腿上，疼得他大叫：“沃森，快来帮我！”。没想到，这句话从电话的这一头传到了那一头，被沃森清楚地听到了。所以，这句话也被作为电话史上的第一句话流传至今。

2．1831年8月：发现电磁感应

1831 年 8 月，英国的法拉第发现了电磁感应现象，麦克斯韦进一步用数学公式阐述了法拉第等人的研究成果，并把电磁感应理论推广到了空间。而 60 多年后，赫兹在实验中证实了电磁波的存在。

电磁波的发现，成为“有线电通信”向“无线电通信”的转折点，也成为整个移动通

信的发源点。正如一位科学家说的那样“手机是踩着电报和电话等的肩膀降生的，没有前人的努力，无线通信无从谈起。”

3．1973年4月：出现第一部移动电话

1973 年 4 月，一名男子站在纽约的街头，掏出一个约有两块砖头大的无线电话，并开始通话，惹得周围人们纷纷关注。这个人就是手机的发明者——马丁·库珀，当时他还是摩托罗拉公司的工程技术人员，而这个无线电话也是世界上第一部移动电话。

4．1983年：第一部真正意义上的手机上市

1983 年，摩托罗拉正式推出了 DynaTAC 8000X，这也是世界上第一台真正意义上的手机。刚上市时，它重达 2.5 磅，也就是 1.2 千克左右，别看它“个头”这么大，真正能支持的通话时间却只有半个小时。那时的它还是名副其实的“大哥大”，零售价高达 3995 美元，在中国黑市被炒到了 5 万元左右。

5．1993年9月18日：中国建成第一个GSM网络

1993 年 9 月 18 日，在浙江嘉兴建成了第一个 GSM 网络。中国移动通讯市场开始了超常规、成倍数、跳跃式的发展，从此移动通讯进入了数字时代。1994 年 10 月，广东数万用户成为第一批 GSM 的使用者，从此正式拉开了中国移动通讯市场高达 3.6 亿用户的序幕。

6．2007年11月5日：Google公司发布Android

2007 年 11 月 5 日，Google 公司发布了基于 Linux 平台的开源手机操作系统——Android。开放手机联盟正式成立，从此掀开了智能手机应用的开发热潮！现在，越来越多的人正享受着智能手机为我们带来的便利和乐趣。

1.1.2　Android 的各个版本

Android 一词的本义为“机器人”，同时也是 Google 公司于 2007 年 11 月 5 日宣布的基于 Linux 平台的开源手机操作系统的名称，该平台由操作系统、中间件、用户界面和应用软件组成，号称是首个为移动终端打造的真正开放和完整的移动软件。图 1.1 为 Android 的 Logo。



图 1.1　Android 的 Logo

到现在为止，Android 已经发布了最新的 4.0 版本，那么历史上 Android 一共有哪些版本呢？作为手机开发人员的你，必定对此有一些了解。现在，让我们重新整理并回顾那些曾经或正在辉煌着的“版本”们，

如表 1-1 所示。

表 1-1　Android各个版本

版　本　号	代　　号	发 布 时 间
Android 1.1		2008 年 9 月
Android 1.5	Cupcake　纸杯蛋糕	2009 年 4 月
Android 1.6	Donut　甜甜圈	2009 年 9 月
Android 2.0	Éclair　松饼	2009 年 10 月 26 日
Android 2.1	Éclair　松饼	2009 年 10 月 26 日
Android 2.2	Froyo　冻酸奶	2010 年 5 月 20 日
Android 2.3	Gingerbread 姜饼	2010 年 12 月 7 日
Android 3.0	Honeycomb 蜂巢	2011 年 2 月 3 日
Android 3.2	Honeycomb 蜂巢	2011 年 7 月 13 日
Android 4.0	IceCreamSandwich　冰激凌三明治	2011 年 10 月 19 日

表 1-1 中列出的仅仅是一些比较重要的版本，诸如 Android 2.1—update、Android 3.1 等一些比较小的更新版本这里就不再罗列。

也许读者朋友们会觉得比较奇怪，为什么 Android 手机版本名称都如此奇怪，实际上如果你仔细观察不难发现：

（1）Android 所有的版本都是以甜点来命名的。

（2）Android 的所有版本的首字母是从 A～Z 排列的。

在笔者成书时，市场最为流行版本是 2.2，本书使用的版本也是 Froyo。目前 Android 最高版本是 4.0，图 1.2 为 Android 4.0 冰激凌三明治的 Logo。



图 1.2　Android 4.0 的 Logo

1.2　开放手机联盟

开放手机联盟的英文全称是 Open Handset Alliance，Android 可以说就是开放手机联盟的成果。它由 Google 公司领导，包括移动运营商、手持设备制造商、零部件制造商、软件

解决方案和平台提供商，以及市场营销公司。

1.2.1　开放手机联盟的目的

开放手机联盟（Open Handset Alliance）是美国 Google 公司于 2007 年 11 月 5 日宣布组建的一个全球性的联盟组织。他们的目标是开发多种技术，大幅削减移动设备和服务的开发和推广成本，从而构建更好的移动电话。

该联盟的创始成员包括 Google、中国移动、T-Mobile、宏达电、高通、摩托罗拉等在内的 34 家行业领头羊。而后不久，华硕电脑、Sharp、华为、海尔、联想、索尼爱立信、爱立信、东芝、中国联通、中国电信、中兴通信等 29 家公司也加入了开放手机联盟，这些公司已经涵盖了全球整个手机产业链。

或许细心的你会发现，手机行业的巨头 Nokia 并没有出现在联盟中，这是因为诺基亚公司一直支持的是其子公司的 Symbian 系统，并且在所有的智能机中都预装了该系统。显然，这和开放手机联盟的初衷背道而驰。

1.2.2　分工合作

从宏观上，一部手机的成型大致可以分为两个步骤：（1）手机制造；（2）软件开发。

1．手机制造

超过一半的开放手机联盟成员是手机制造商，如三星、HTC、LG 以及最近开始涉足手机制造业的华为等。当然制造商中还包括了很多卓越的半导体制造商，如英特尔、高通以及德州仪器等。

这些公司通力合作一起设计完成了第一部 Android 手机——T-Mobile G1。该手机由宏达电子（HTC）设计开发，并由 T-Mobile 提供配套服务，于 2008 年 9 月 23 日在美国正式上市。

2．软件开发

软件开发正是本书讨论的重点，在本书中我们将学习如何使用 Android SDK 开发运行在 Android 手机上的应用程序。我们的最终目标是能够自己开发出任何你希望获得的功能，当然这也许需要一些硬件支持。

就目前来看，你可以到 Android Market 中下载你需要的程序。如果你对 Android Market 还不是很了解，没有关系，下一节中我们将为你介绍。现在的 Android Market 中已经包含了上万种应用，不管是实用工具、教育、影音还是游戏社交，包罗万象、应有尽有。

1.3　Android 中的个人英雄主义

Android 是一个完全整合的移动软件系统，包括操作系统、中间件、便于用户使用的界面以及各类应用。为了鼓励开发人员参与到 Android 中，Google 举办了两届开发者挑战

赛，每次大赛的总奖金都达到了 1000 万美元。

1.3.1　第一届挑战赛冠军介绍

第一届开发者大赛收到了 1788 件参赛作品，其中包括了各个方面：游戏、LBS（Location Based Service，定位服务软件）、各类使用工具以及软件交友平台等。

想要成功创作出一个好的作品也许可以从以下几点考虑：

- 巧思
- 实用
- 完善
- 美观

巧思就是创意，一个创意能得到什么？我们无法估量，在软件开发中，一个创意也许是一个成功案例的开始，而一个完美的案例也许又会给你带来一个机遇，一个机遇或许就改变了你的现状，改变了你的人生。作为一个程序员千万不要被条条框框束缚了自己的思想，学会发散性思维也许就是你的第一课。

实用是指你的软件到底能为用户做些什么？是否能够切实为用户带来便利和享受？学会多从用户的角度出发看待问题，不要只从简单的开发来考量。

举一个最简单的例子：在 Android 开发中有两种组件，一个是编辑框，另一个是单选框，这两种组件在第 5 章中都会有详细的讲解，这里读者只要知道它们就可以了。

编辑框顾名思义就是一个可以用来输入信息的窗口，单选框则是提供多个选项但只能选择其中之一的一类组件。当你希望用户输入性别时你可以选择提供一个编辑框让用户输入，也可以提供一个单选框，在其中设置“男”和“女”两个选项。从开发的角度来说肯定是第二种方案较第一种麻烦，但从用户的角度来说，必定是第二种方法更实用、更贴心。所以毋庸置疑，如果你希望你的软件更实用的话，乖乖地选择第二种吧！

完善则是指你的软件是不是仅仅只是实现基本功能？是不是还有更多的附加功能可以添加。再举个简单的例子：一个软件提供了列车的查询功能，要实现它肯定非常简单，只需到网上找到一个信息提供商，然后获取信息并列表显示就可以了。可是仅仅如此么？这一大堆的数据列在用户的面前是不是让用户觉得老虎咬刺猬——无从下口呢？为了让用户使用更方便，我们可以提供按照日期查询、按照班次查询、按发站/到站时间查询、按车辆类型查询等种种功能。这样查询列车的功能就比较完善了，用户使用起来会更方便、更贴心。

美观同样是评定一个软件优秀与否的重要标准，在这个追求个性的年代，绚丽的界面和强大的功能一样重要。一个好的界面会起到“先声夺人”的作用。设想一下，同样的功能，不同的界面，用户肯定会选择看起来更舒服的那一款。

以下是第一届 Android 开发挑战赛的冠军——GoCart，希望读者朋友们可以从该作品中得到启发。GoCart 是一款掌上移动购物的应用程序，它大致上分为 4 种功能：

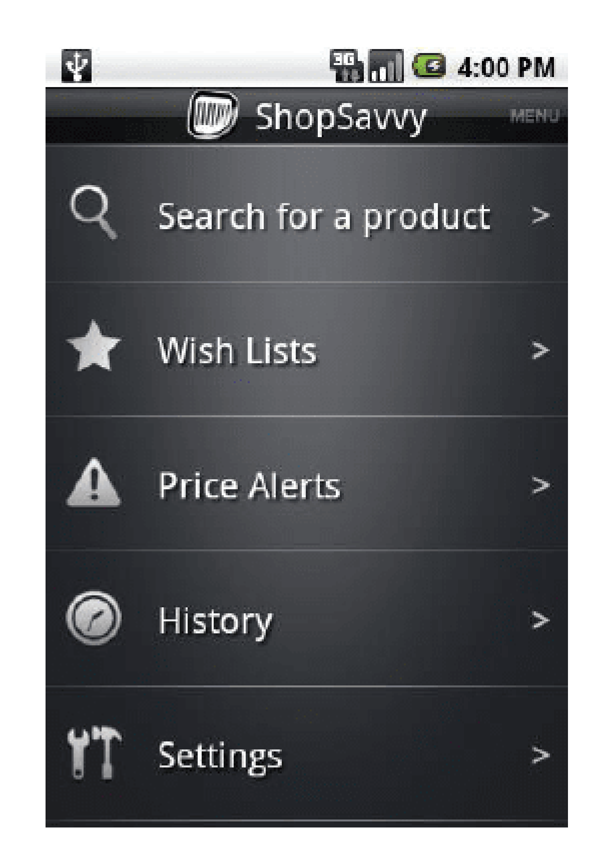
（1）获得商品信息。该功能的实现需要使用到 Android 手机的摄像头，通过摄像头扫描商品的条形码可以获得其具体信息。

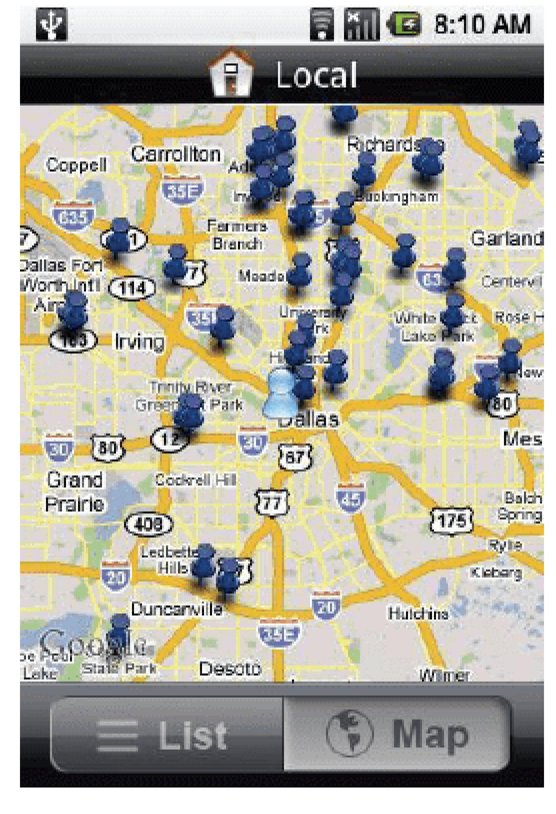
（2）扫描成功后，GoCart 将会使用 Android 的网络功能，在网上商店查找这个商品的价格，从而进行网上购物。

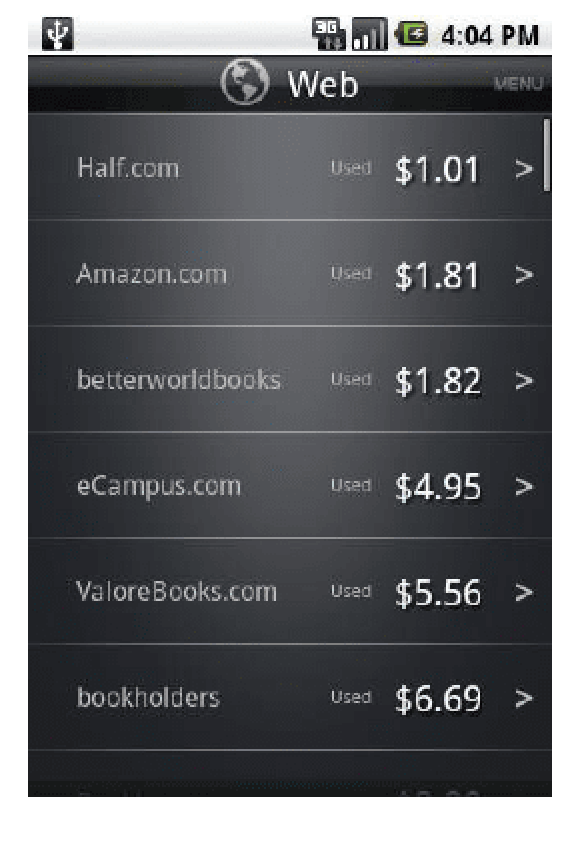
（3）如果你不相信网上购物，你也可以使用 GoCart 的 GPS 功能，查询你现在所在的位置，然后确定你附近有哪些商店有该类产品，如果信息充分，甚至可以查询商店的库存，以确定是否有必要前往该商店。

（4）如果你还是不满意，没有关系，你可以在 GoCart 中设定一个期望价格区间，这样一旦网上有符合你条件的商品信息出现，GoCart 会第一时间提醒你。

以下是 GoCart 的使用截图，如图 1.3 所示。







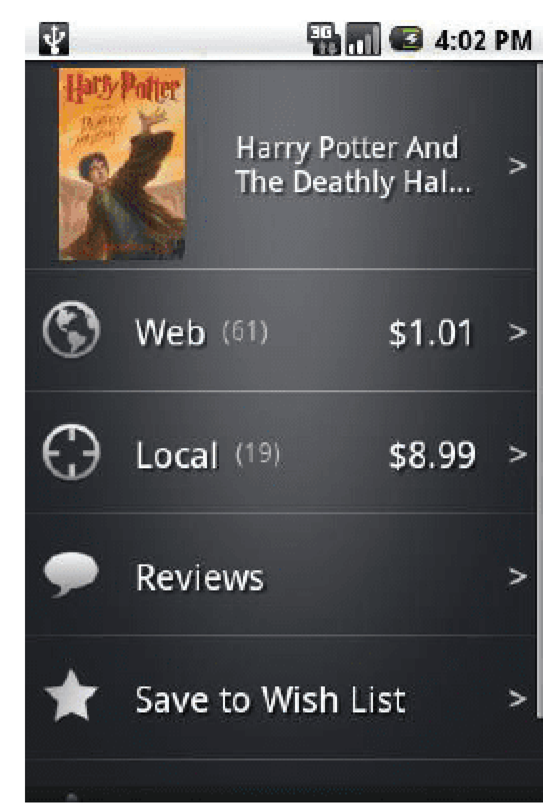


图 1.3　GoCart 使用截图

1.3.2　第二届挑战赛冠军介绍

继第一届 Android 开发者挑战赛成功举办后，Google 又举办了第二届挑战赛，获得本次比赛冠军的是一款叫做 SweetDreams（甜梦）的软件。该应用的主要功能是：

（1）在自己入睡之后，自动将某些人的来电转为语音邮件，从而避免了不必要的骚扰。

（2）该应用同样可以设置手机的蓝牙、屏幕、WIFI 等设备的开关，从而帮助手机节省电能。

以下是甜梦的使用界面截图，如图 1.4 所示，不得不说，它的界面的确简洁美观，功能也非常强大，当然最重要的还是它的创意。

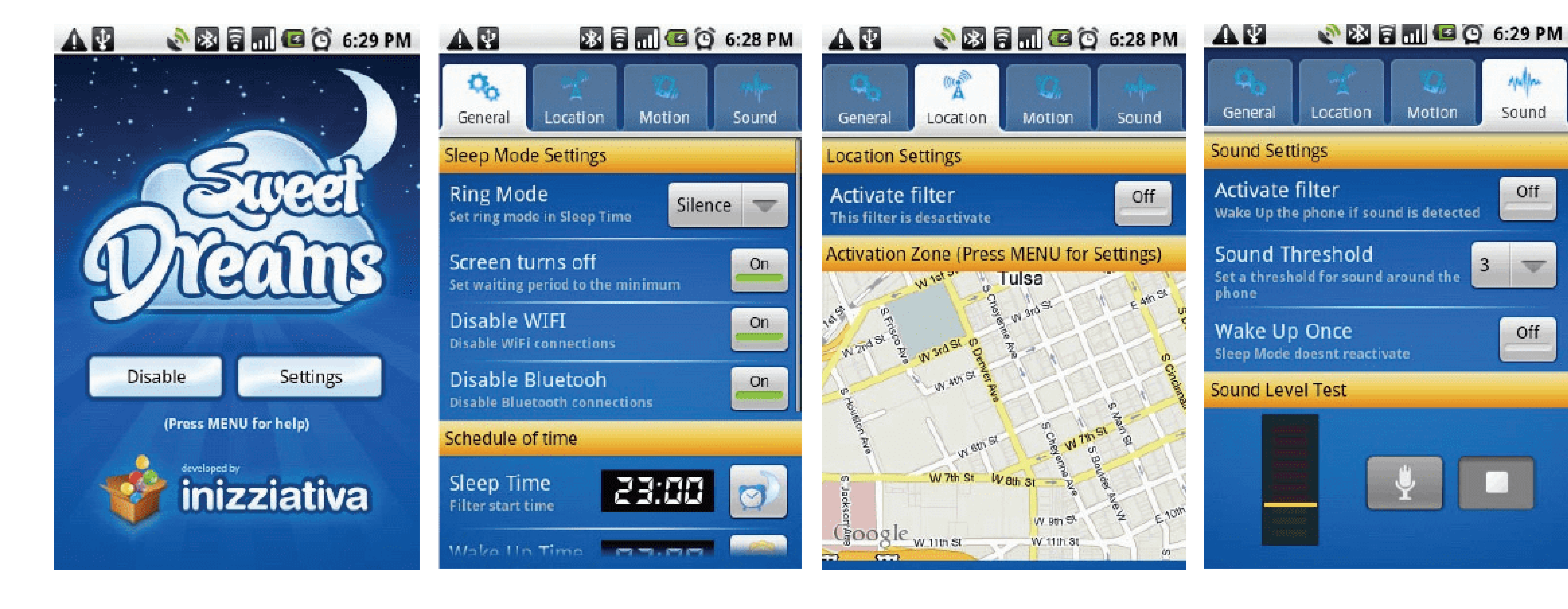


图 1.4　“甜梦”使用截图

1.3.3　Android Market

手机市场长期以来存在着若干问题，如：

- 同类竞争软件的数量限制
- 价格限制
- 盈利模式限制
- 客户群大小的限制

以上的 4 个问题只是在 Android 出现之前在手机开发市场上比较常见的一些问题，还有更多的问题这里并没有列举。可是这些长期存在的问题就没有一个好的办法解决吗？答案当然是否定的。

Android Market 是 Google 开发的一款移动应用商店，在该商店中你可以自由地发布任何一款软件而无需验证。不管是免费软件还是收费软件、共享软件还是测试软件，只要它可以成功运行，你都可以将它无障碍地发布到 Android Market 中。

这就巧妙地解决了上述的难题，如第 4 点——客户群大小的限制。以往的手机市场中，运营商期待赚取大额利润所以往往拒绝为小客户群体开发软件。现在有了 Android 市场，这一切都发生了改变。这就像是一条广告语：只有想不到，没有做不到。事实上，在 Android Market 中已经有上万种软件可供选择，每时每刻都会有新的软件被提交给用户使用。图 1.5 是 Android Market 的 Logo。



图 1.5　Android Market 的 Logo

1.4　Android 平台

从宏观上说 Android 是一个完整、开放而又免费的移动平台。它完整是因为设计人员在开发之初就综合考虑了方方面面：从一个安全的操作系统出发，构建一个完整的应用程序框架，从而开发出各类健壮的应用程序；它开放是因为 Android 公布了它的所有源代码，这样开发人员就可以很方便地访问手机的各类设备；它免费是指在该平台上开发软件，无论是开发工具还是签名认证都是免费的，你无需担心任何版权支出。从狭义上，Android 是一个操作系统，构建于 Linux 操作系统的基础上，安全、可靠而又高效。

1.4.1　Android 体系结构

Android 系统自上而下共有 4 层：

- 应用层——Applications
- 应用框架层——Application Framework
- 核心库和运行时环境层——Libraries 和 Android Runtime
- 操作系统层——Linux Kernel

图 1.6 形象地说明了 Android 的体系结构，接下来我们就围绕这张图对 Android 系统进

行较为深入的讲解。



图 1.6　Android 系统结构图

1．应用层

在应用层中我们可以使用 Java 语言进行各种应用程序的开发。包括桌面、联系人、电话、浏览器、电子邮件客户端、SMS 程序、日历、地图等各种功能。该层也是本书重点讨论的一层。

2．应用框架层

该层为系统提供了各种各样的 API，它包括：

（1）Activity Manager：活动管理器，一个应用程序由至少一个活动（Activity）构成，活动管理器负责管理 Activity 的生命周期，并为程序提供退出机制。

（2）Window Manager：窗口管理器，管理所有的窗口程序。

（3）Content Providers：内容提供者，负责共享程序的数据，该机制解决了各个应用程序的数据私有和共享的问题。在本书第 9 章会有详细讲解。

（4）View System：视图系统，可以用来构建应用程序，它包括各种可重用的组件：列表、网格、文本框、按钮等等。

（5）Notification Manager：消息管理器，它可以帮助开发者在状态栏中显示自定义的提示信息。

（6）Package Manager：包管理器，它可以帮助开发人员管理所有的包。

（7）Telephony Manager：电话管理器，管理 Android 手机中所有的电话接入和拨出等

操作。

（8）Resource Manager：资源管理器，提供非代码资源的访问，如本地字符串、图形和布局文件。

（9）Location Manager：位置管理器，使用它可以开发 LBS（Location Based Service）程序。

（10）XMPP Service：可扩展通讯和表示协议服务（XMPP The Extensible Messaging and Presence Protocol），XMPP 是一种基于 XML 的协议，具有超强的可扩展性。经过扩展以后的 XMPP 可以通过发送扩展的信息来处理用户的需求。

3．各种库和运行时环境层

Android 应用框架需要系统底层的一些 C/C++库的支持。这些库包括：

（1）Bionic 系统 C 库：C 语言标准库，系统最底层的的库，C 库通过 Linux 系统来调用。

（2）多媒体库：Android 系统多媒体库，基于 PackerVideo OpenCORE，支持各类音频格式的录制和播放，包括：MPEG4、MP3、AAC、AMR 等；支持各类视频格式的录制和播放：包括 3GP、MP4 等；支持各类图片格式的处理，包括：JPG、PNG 等。

（3）SGL：2D 图形引擎库。

（4）SSL：位于 TCP/IP 协议与各种应用层协议之间，为数据通信提供支持。

（5）OpenGL ES 1.0：支持 3D 效果。

（6）SQLite：关系数据库，提供数据存储服务。

（7）Webkit：Web 浏览器引擎。

（8）FreeType：提供位图和矢量的支持。

在 Android 操作系统中，每个 Java 程序都运行在一个独立的 Dalvik 虚拟机上。Dalvik 被设计为一个设备，可同时高效地运行多个虚拟系统。每一个 Android 应用都运行在一个 Dalvik 虚拟机实例中，每一个虚拟机实例都是一个独立的进程空间。它只能执行.dex 的可执行文件，也就是说当 Java 程序通过编译后，生成.class 文件，最后还需要通过 SDK 中的 dx 工具转为成.dex 格式才能正常地在虚拟机上执行。

4．操作系统层

（1）显示驱动（Display Driver）：基于 Linux 的帧缓冲（Frame Buffer）驱动。

（2）键盘驱动（KeyBoard Driver）：作为输入设备的键盘驱动。

（3）USB 驱动（USB Driver）：为设备提供 USB 驱动。

（4）Flash 内存驱动（Flash Memory Driver）：闪存驱动程序。

（5）照相机驱动（Camera Driver）：常用的基于 Linux 的 v412（Video for Linux）的驱动。

（6）音频驱动（Audio Driver）：常用的基于 ALSA 的高级 Linux 声音体系驱动。

（7）蓝牙驱动（Bluetooth Driver）：基于 IEEE 802.15.1 标准的无线传输技术。

（8）WiFi 驱动：基于 IEEE 802.11 标准的驱动程序。

（9）Binder IPC 驱动：Android 的一个特殊的驱动程序，提供进程间通信的功能。

（10）Power Management（电源管理）：管理电池电量。

1.4.2　熟悉的开发工具

大致了解了 Android 系统的体系结构以后，接下来需要为我们的应用开发做准备了：首先我们需要一个称手的开发工具。

Google 官方推荐我们使用 Eclipse 作为 IDE 来辅助开发。为什么这么多的 IDE 中我们选择了 Eclipse 呢？首先，Eclipse 是完全免费的；其次，Eclipse 是目前最流行的 Java 开发 IDE，它可以装插件，无限地增强功能；最后，Google 已经为 Eclipse 开发了一款 ADT（Android Develop Tools）。使用 Eclipse+ADT 可以很方便地创建和编译 Android 工程，从而避免了复杂的命令行模式。

本书第 2 章中我们会详细讨论 Windows XP + Eclipse + ADT 的开发环境的配置。图 1.7 和图 1.8 分别是 Eclipse 启动和运行时的截图。



图 1.7　Eclipse 启动

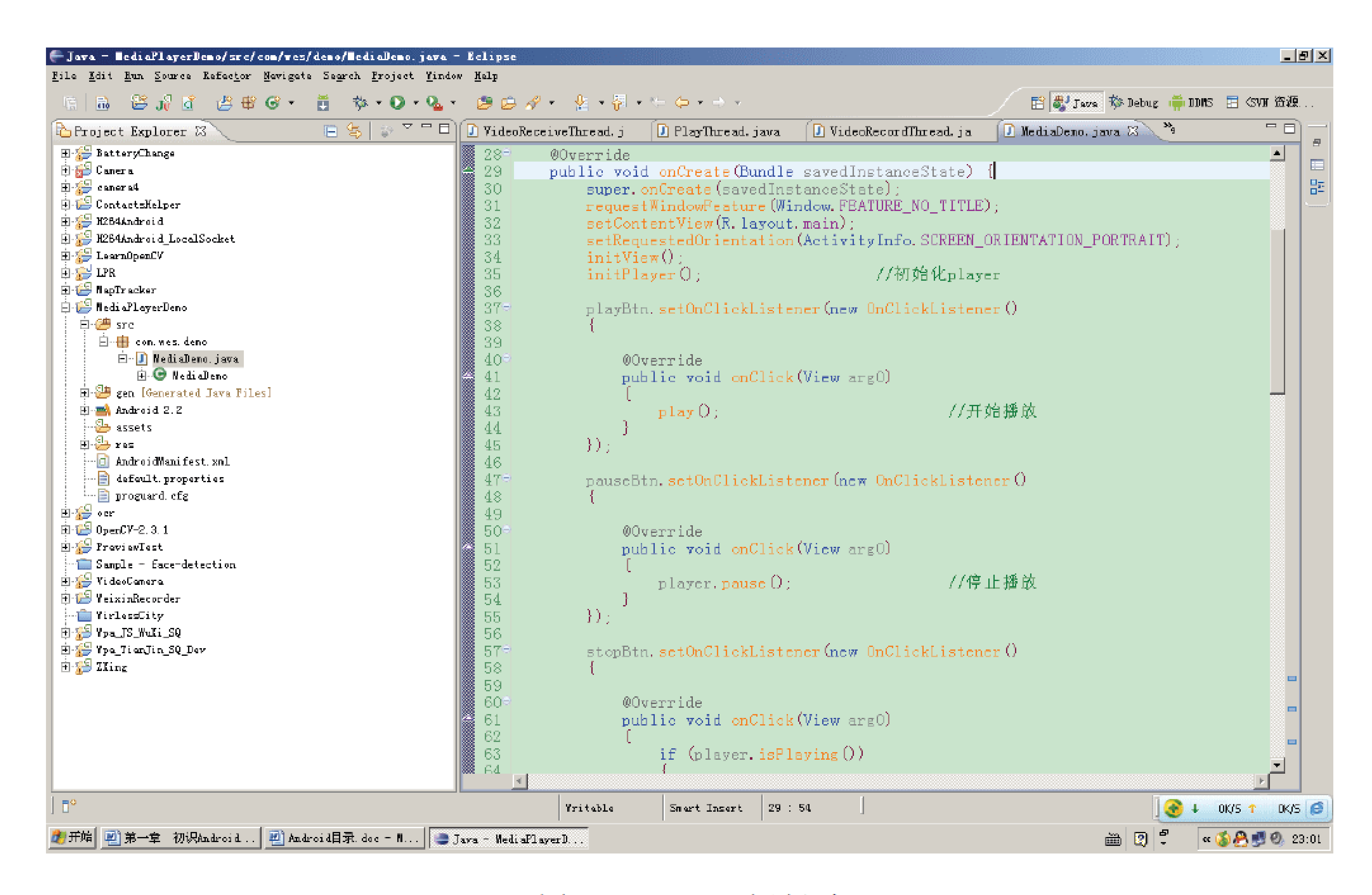


图 1.8　Eclipse 运行时

1.4.3　合理的学习曲线

在 1.4.2 节我们初步认识了以后我们要使用的开发工具，可是“工具”再好，那也仅仅是个工具，最重要的还是我们的“内功”。本小节就讨论应该怎样修炼好 Android 开发的“内功”。怎样学习 Android 才能更快速地上手呢？当然，在快速上手的同时还需兼顾内容的完整性以及一定的深入性。为此，本节特别制定了一条合理的学习曲线，如图 1.9 所示。

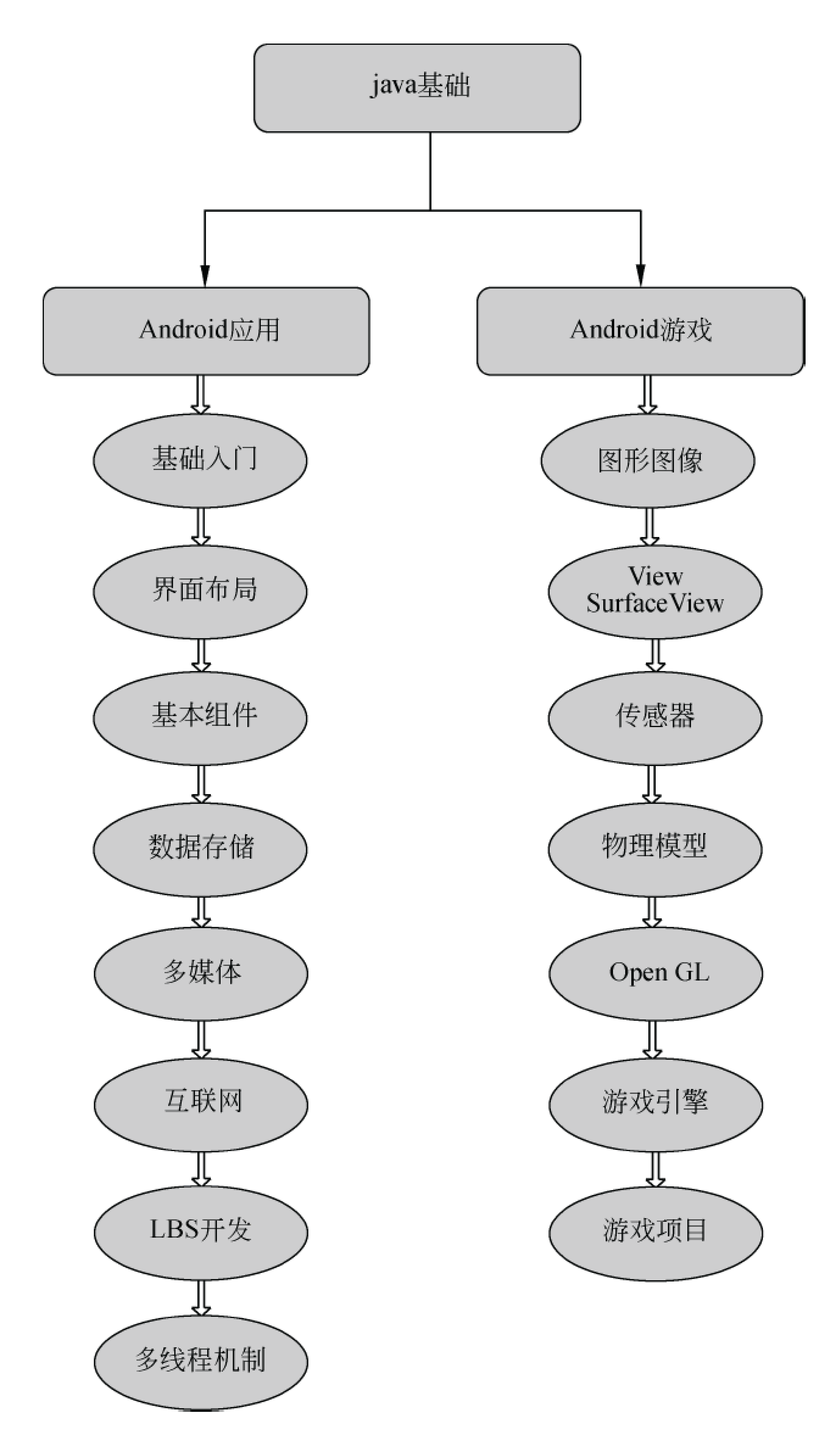


图 1.9　合理的 Android 学习曲线

由于本书侧重讲解 Android 的应用开发，所以对 Java 以及 Android 游戏等方面的知识讲解较少。在 Android 应用开发中，各个知识点的粗略介绍如表 1-2 所示。

表 1-2　Android应用开发中知识点的介绍

知　识　点	介　　绍
Android 基础入门	搭建 Android 开发环境并创建第一个 Android 程序——HelloWorld
Android 界面布局	使用界面的各种布局类，如 LinearLayout、FrameLayout、TableLayout、AbsoluteLayout、RelactiveLayout 以及各种 View 的子类，如 TextView、EditText、Button 等
Android 基本组件	使用 Activity、Intent、BroadcastReceiver、ContentProvider 等
Android 数据存储	使用 Android 中的数据存储，包括 SharedPreferences 以及 SQLite3 数据库以及文件存储等
Android 多媒体	使用多媒体框架进行拍照、音频的录制和播放、视频的录制和播放等
Android 网络开发	使用网络接口 HttpUrlConnection、HttpClient 等
Android 中的 LBS 开发	使用 Google 地图开发接口实现地图展示、定位、查询等功能
Android 多线程	使用多线程机制，熟悉 Android 平台的消息处理机制，掌握在非主线程中更新界面
Android 应用	综合应用以上知识点开发出使用的应用程序

1.5　小　　结

Android 作为新一代的移动开发平台，是手机行业发展的必然。它的出现打破了传统的移动开发格局，让我们开发人员可以轻松地投入到 Android 开发潮流中来。本章首先回顾了手机发展的里程碑，接着讲解了开放手机联盟的成立，然后重点讲解了 Android 平台的特点以及 Android 学习曲线。在下一章，我们将认识并使用 Android 开发的“四大法宝”，通过它们进行 Android 开发环境的配置。

第 2 章　搭建你的开发环境

工欲善其事，必先利其器，在开始 Android 学习之旅前，我们首先要“磨”好我们要使用的“刀”，才能更有效地完成“砍柴工”。本章将带领大家完成 Windows 环境下 Android 开发环境的搭建，并解析每个步骤的具体含义。

2.1　配置前的准备工作

开始配置开发环境之前，我们要实现准备好开发的“四大法宝”：Java 开发工具包 JDK、Eclipse 开发环境、Android 软件开发包 Android SDK 以及 Android 开发工具包 ADT。当然这些都是免费的！

2.1.1　Android 支持的操作系统

Android 开发环境需要被搭载于一个操作系统上，在以下的操作系统中可以进行 Android 开发：

- Windows XP 或 Vista
- Mac OS X 10.4.8 及以上
- Linux Ubuntu Drapper

选择一个你最熟悉的操作系统进行环境搭建。由于篇幅限制，本书仅以 Windows XP 下的开发环境配置为例，其他操作系统的配置步骤类似。

2.1.2　准备“四大法宝”

1．准备JDK

由于 Android 应用开发使用了 Java 语言，所以我们必须安装 Java 开发工具包——JDK（Java Development Kit）。有过 Java 开发经验的读者应该知道该包的用处，这里只做简单介绍：JDK 是 Java 的核心，包括 Java 运行环境、Java 工具和 Java 基础的类库。没有安装 JDK 就无法安装 Eclipse 开发平台。

Android 开发环境需要的 JDK 版本是 JDK SE 1.7 及以上，下载地址为：http://www.oracle.com/technetwork/java/javase/downloads/index.html，如图 2.1 所示。

选择下载 JDK 就可以，不需要 JRE。单击 Download 后，选择合适自己的操作系统的 JDK 进行下载。本书使用的是 JDK 版本是 jdk-6u27-windows-i586。

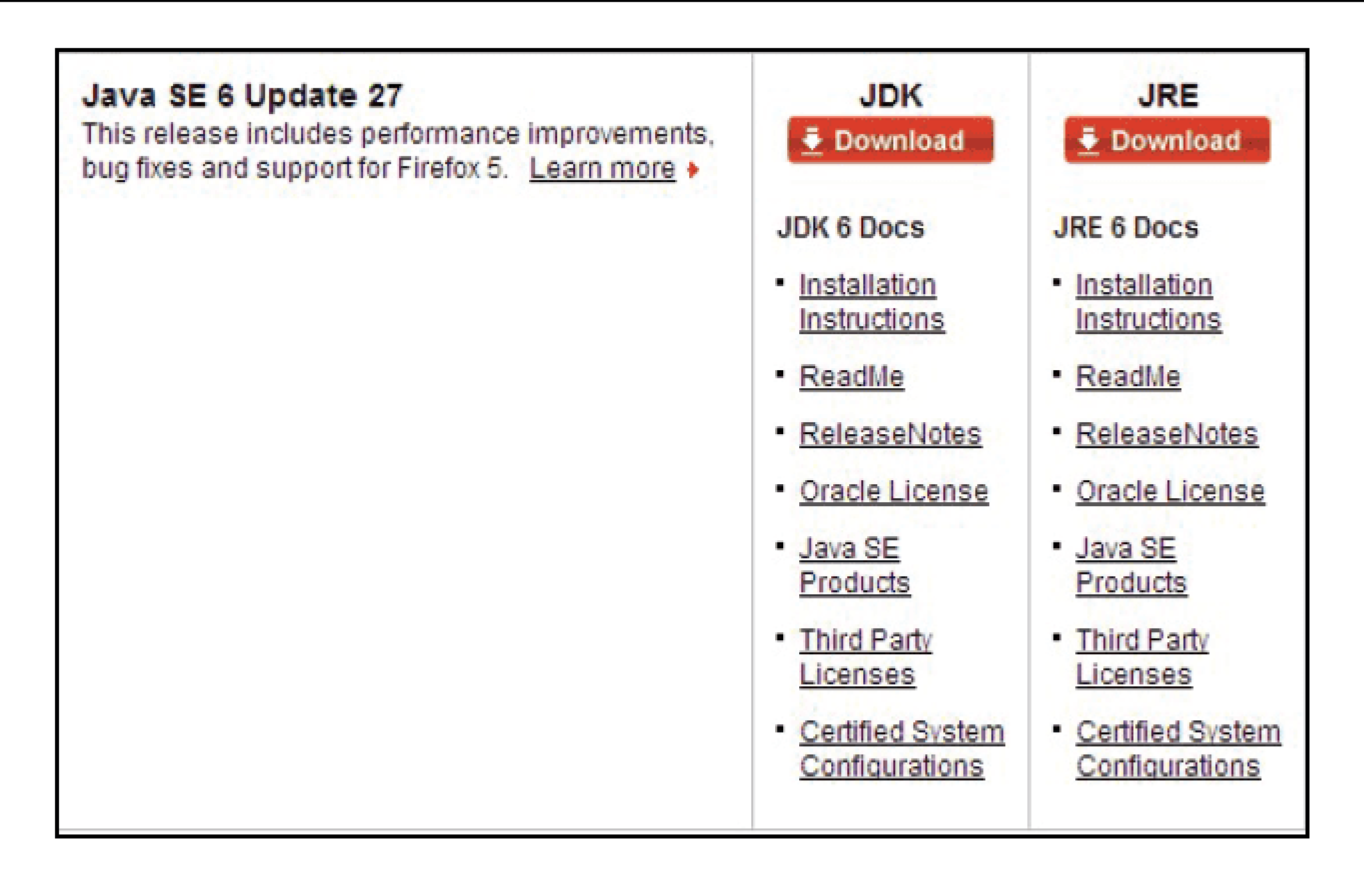


图 2.1　JDK 下载页面

2. 准备Eclipse

大多数的开发人员选择时下流行的 Eclipse 集成开发环境进行 Android 的开发。Eclipse 是一种基于 Java 的可扩展开源开发平台，就其自身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境。Eclipse 下载地址为：www.eclipse.org/downloads，如图 2.2 所示。



图 2.2　Eclipse 下载页面

选择第一个选项——Eclipse IDE for Java Developers，并选择合适的版本进行下载，本机使用的是 Windows 32Bit。

3. 准备Android SDK

要进行 Android 开发必然要 Android SDK，也就是 Android 专属的软件开发包。下载地址是：http://developer.android.com/sdk/index.html，如图 2.3 所示。

Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r13-windows.zip	36487911 bytes	de8a039891e5e65b7742f188f07b992d
	installer_r13-windows.exe (Recommended)	36533357 bytes	cd3a76fe2b8ed62b2d03cf1851692e2d
Mac OS X (intel)	android-sdk_r13-mac_x86.zip	30233944 bytes	f4002a0344b48856c09dec796acecd4d
Linux (i386)	android-sdk_r13-linux_x86.tgz	30034328 bytes	d80d7530a46c665644ae76084a9a0dc4

图 2.3　Android SDK 下载页面

选择 android-sdk_r13-windows.zip 进行下载。事实上，我们下载的并不是 Android SDK，而是一个 SDK 的下载安装器。

ADT 不需要准备，在后面的安装过程中可以直接下载并安装。

2.2　安装并配置 JDK

上面已经介绍了下载的地址了，如何安装呢？本节就来介绍安装及其配置 JDK。

2.2.1　安装 JDK

下载完成后，我们已经准备好了如图 2.4 所示的安装包。



图 2.4　准备好的安装包

首先我们开始安装 JDK，双击 jdk-6u25-windows-i586.exe 开始安装，首先出现如图 2.5 所示的安装界面。

（1）单击“下一步”按钮后进入设置界面，单击“更改”按钮选择你希望安装的位置，如图 2.6 所示。



图 2.5　JDK 安装向导　　　　图 2.6　安装 JDK

（2）继续单击“下一步”按钮，安装 JRE（Java Runtime Environment），同样单击“更改”按钮可以选择 JRE 的安装位置，如图 2.7 所示。

（3）继续单击“下一步”按钮，开始安装，如图 2.8 所示。

（4）安装完成后，出现如图 2.9 所示的提示则表示安装成功。



图 2.7　安装 JRE　　　　图 2.8　正在安装



图 2.9　安装完成

2.2.2　配置 JDK

安装完成后我们还需要对 JDK 进行一些配置，这里的配置指的是设置相应的环境变量。首先打开“我的电脑”，选择“属性”，选择“高级”标签，打开如图 2.10 所示的设置界面。

单击“环境变量”按钮后进入图 2.11 所示的环境变量设置界面。

接下来开始进入具体的配置。

1．配置JAVA_HOME变量

单击“新建”按钮，注意是“系统变量”下的“新建”按钮而不是“用户变量”下的“新建”按钮。在弹出来的对话框中，在“变量名”中输入 JAVA_HOME，在“变量值”中输入 JDK 的安装路径，这里的安装路径为 C:\Program Files\Java\jdk1.5.0_07，如图 2.12 所示。

接下来让我们来分析一下为什么要配置该变量。市场上有很多书只是告诉读者朋友们完成了环境的配置，但是却没有告诉读者为什么要这么做。这样对于开发人员是很不利的，

试想如果连你的工具你都不够了解，那么谈何利用它开发呢？

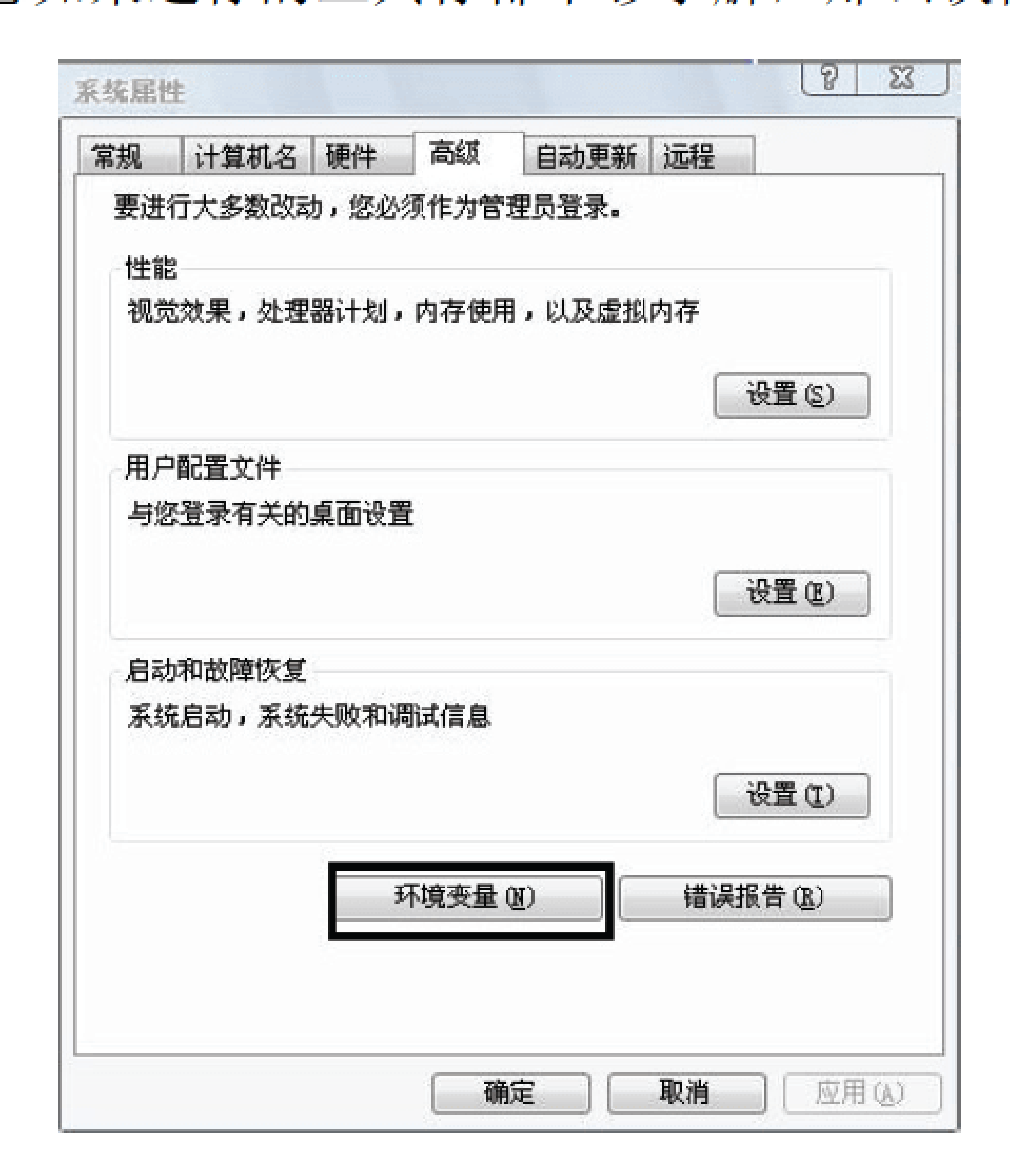




图 2.10　“高级”选项卡



图 2.11　环境设置界面

我们配置该变量的目的是得到 JDK 的地址引用，以后需要使用 JDK 的路径时只需输入%JAVA_HOME%就可以了，这样避免了每次都输入长长的路径，从而减少了错误的产生。而且以后 JDK 的路径发生改变时只要修改这一处就可以了，不需要到处寻找哪里使用了 JDK 的路径，最后有些应用需要使用该变量，如果没有该 JAVA_HOME 变量，软件就无法使用。

2. 配置Path变量

选中图 2.11 中标注出来的变量，然后单击“编辑”按钮，弹出如图 2.13 所示的对话框：

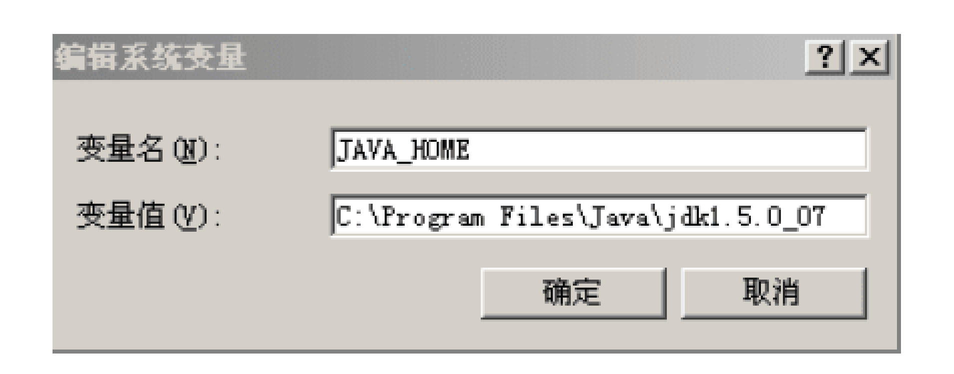


图 2.12　JAVA_HOME 配置



图 2.13　配置 Path 变量

注意不要删除其他的相关信息，在所有信息的最末尾处添加一个分号，将之前安装的 JDK 的 bin 目录填写上去。这个时候就需要使用刚才配置的 JAVA_HOME 变量了。添加的信息如下所示：

```
;%JAVA_HOME%\bin
```

是不是很方便，如果没有设置 JAVA_HOME 变量，我们就要输入 C:\Program Files\Java\jdk1.5.0_07\bin 了。输入完成后单击“确定”按钮。

那么我们又为什么要配置 Path 变量呢？因为系统将根据该 Path 变量找到 Java 编程中需要使用的一些程序，如 javac.exe 以及 java.exe。javac.exe 用于编译 Java 源代码，产生.class 文件，java.exe 则是用于执行编译过后的.class 文件。

3．配置CLASSPATH变量

在图 2.11 中的系统变量下单击“新建”按钮，在弹出的对话框中的变量名中输入 CLASSPATH，在变量值中输入如下的代码，如图 2.14 所示。

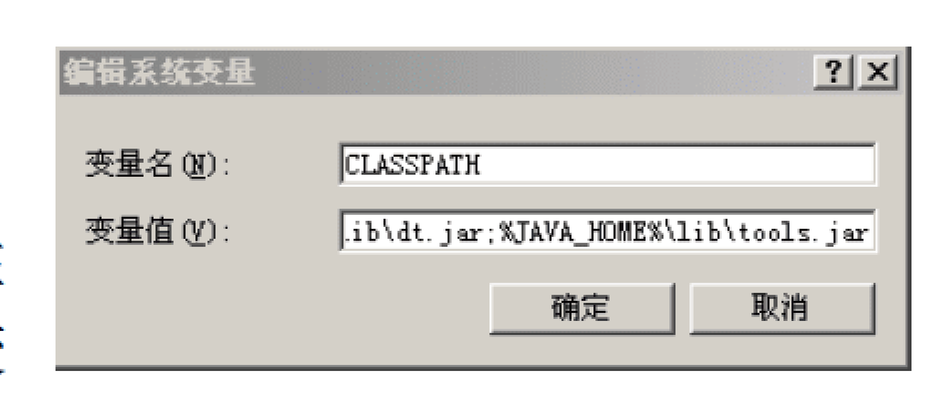


图 2.14　配置 CLASSPATH 变量

```
.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar
```

注意这里的变量值中首先要输入“.;”该“.”表示当前路径。

配置该变量的意义在于：

（1）告诉 Java 解释器到哪里去找标准类库。标准类库是其他开发者已经写完成的，供其他人使用的.jar 文件，例如我们常用到 java.lang 包。这些标准类库就保存在刚才我们输入的两个.jar 文件中。

（2）配置到这里.JDK 的环境变量就设置完成了，接下来我们赶紧验证一下，是不是真的生效了呢？

在系统的 DOS 命令行窗口中输入：

```
java -version
```

如果弹出如图 2.15 所示的信息表示安装成功，如果没有成功则表示配置没有生效，请仔细查看是不是有输入错误。

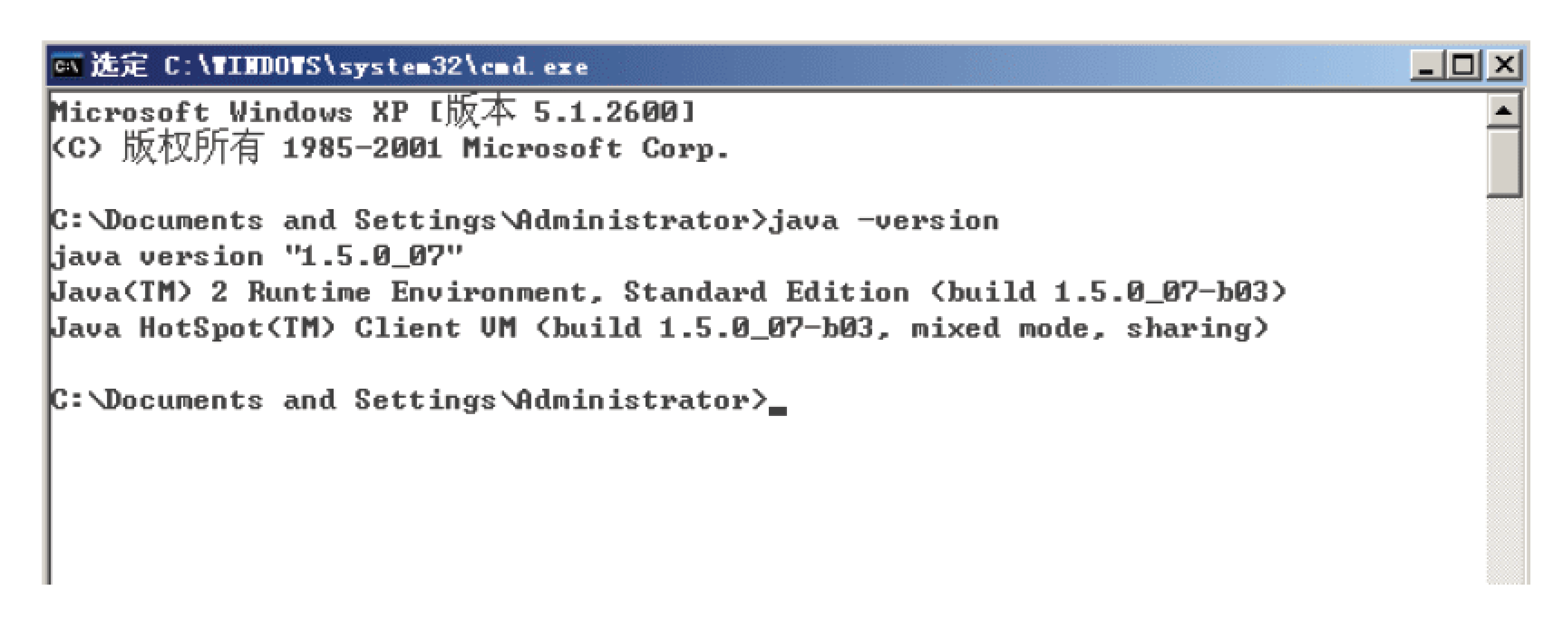


图 2.15　验证配置成功

到这里 JDK 就配置完成了，接下来进行 Eclipse 的安装与配置。

2.3　安装并配置 Eclipse

Eclipse 实际上并不需要安装，解压我们之前下载的安装包，找到其中的 eclipse.exe 文件，双击打开就可以了。

2.3.1　运行 Eclipse

运行后进入开始画面，如图 2.16 所示。

第一次打开的时候会提醒用户设置 workSpace，workSpace 就是你工作的目录，如图 2.17 所示。以后新建的工程会保存在该 workSpace 中。

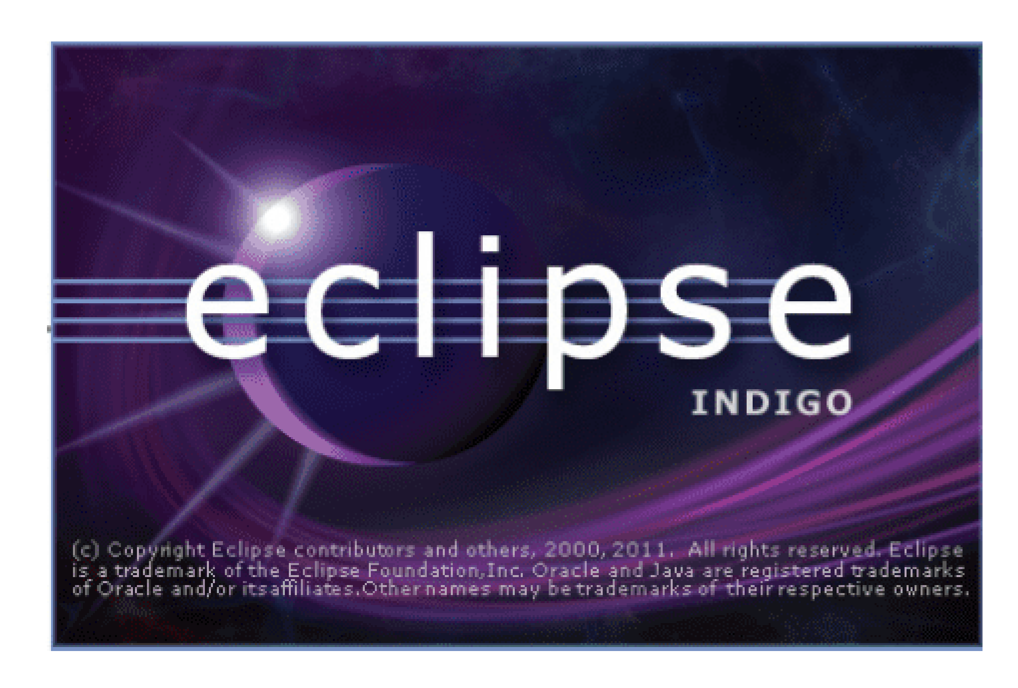


图 2.16　Eclipse 初始化界面

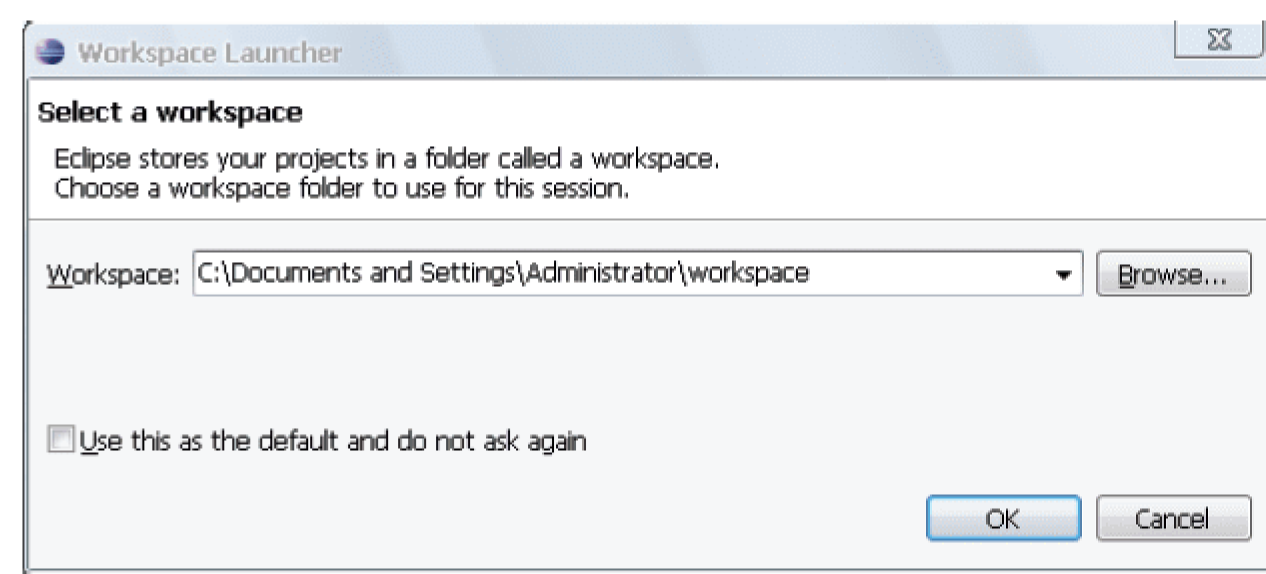


图 2.17　选择 workSpace

初始化成功后进入欢迎界面，如图 2.18 所示。



图 2.18　欢迎界面

单击最右边的箭头图标进入工作环境，当然也可以单击其他图标浏览相关信息。

2.3.2　了解 Eclipse

Eclipse 第一次进入编程界面时如图 2.19 所示。如果读者还没有用过 Eclipse，不要着急，在之后的讲解中，Eclipse 的使用技巧会穿插给出。这里首先对其主要的结构进行简单的介绍。

① 包浏览器：在包浏览器中可以查看工程的层次结构，同样可以进行新建、修改、打开、关闭工程等一系列操作。

② 代码编辑框：在这里开发人员进行具体的代码的编辑和修改。

③ 任务列表：这里罗列出一些执行的任务，显示相关信息，如是否正在执行等。

④ 大纲：这里显示了该类包含的变量名、方法名以及直接的层次结构等。

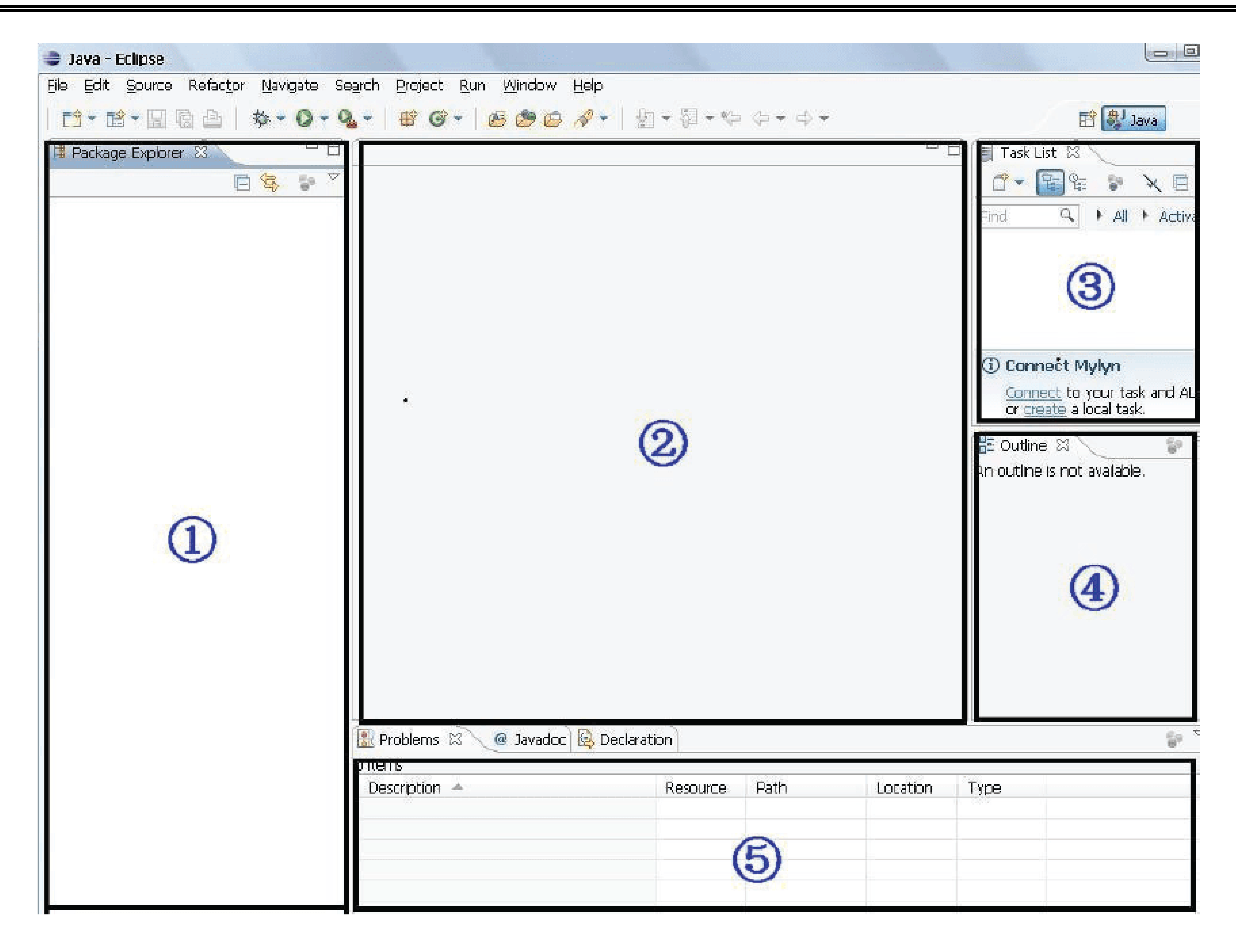


图 2.19　Eclipse 工作环境

⑤ 问题显示栏：运行时遇到的错误在这里显示，主要用于调试。当然在安装了 Android SDK 以后，可以选择查看 Logcat 更有效地追踪问题。

在 Eclipse 界面的第一行有很多菜单，这些菜单的功能在以后的讲解和使用中我们会逐步学习。界面第二行的快捷方式的使用同样会在以后学习。

2.4　安装并配置 Android SDK

通过以上的安装，读者已经可以正常地运行 Java 了，但是要进行 Android 开发显然是不够的，要进行 Android 开发我们还需要安装 Android SDK。在 2.1 节中我们已经准备好了 Android SDK 的下载工具，本节将讲解如何通过该工具安装 Android SDK。

2.4.1　下载 Android SDK

解压缩之前下载的 android-sdk_r13-windows.zip 文件，进入层级目录，显示如图 2.20 所示。

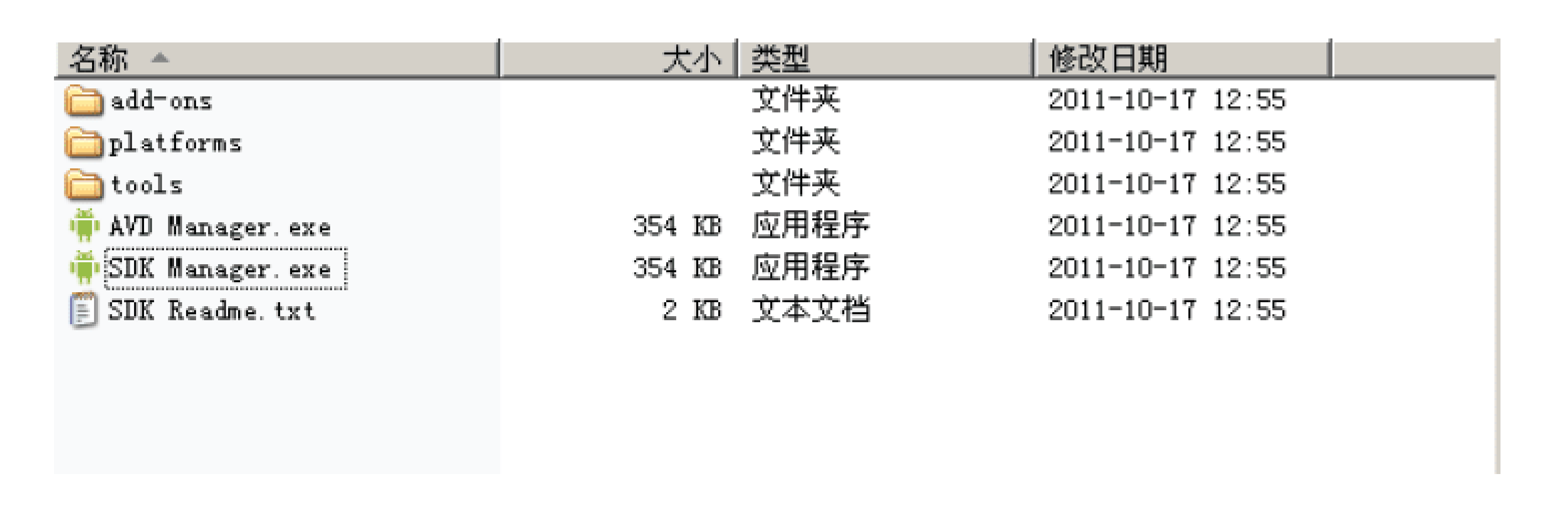


图 2.20　SDK 管理器

运行 SDK Manager.exe，进入界面，如图 2.21 所示。

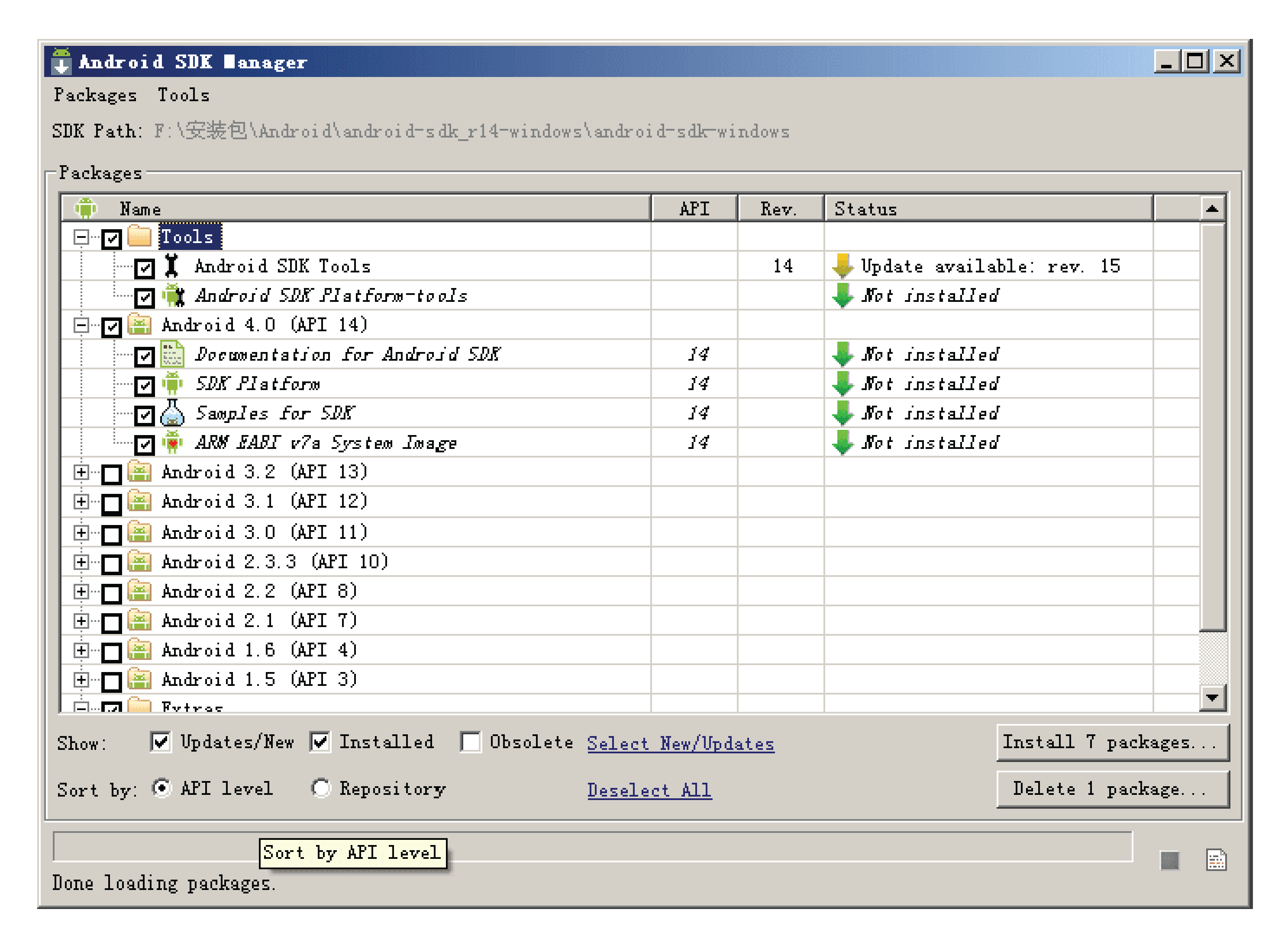


图 2.21　选择下载的 SDK 版本

本书使用的 SDK 版本为 Android 2.2（API 8），因为它依然是目前市场占有率最高的 Android 版本。当然你也可以选择更加高级的版本下载。勾选了你希望下载的版本后，单击 Install packages 按钮，弹出对话框，如图 2.22 所示。

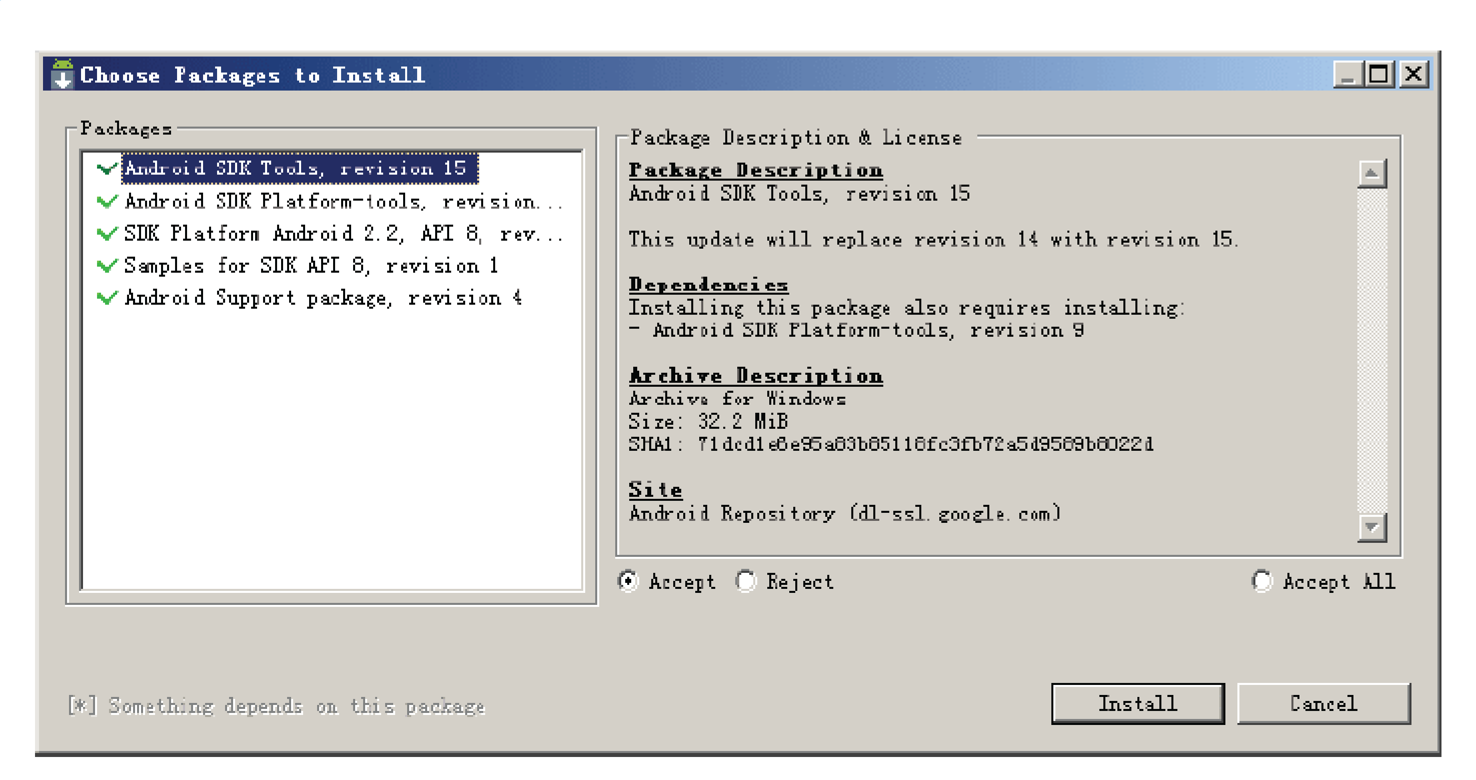


图 2.22　下载 SDK

单击 Install 按钮，开始下载，这可能会花费你一些时间，不要着急，你可以去做一些其他的事情。

2.4.2　配置 SDK

下载完毕之后，与 JDK 一样，我们还需要对其进行一些配置工作。打开环境变量的设置窗口，在 Path 变量后添加 Android SDK 下的 tools 文件夹的路径。这里的路径为：

```
D:\android\android-sdk-windows\tools
```

具体的配置过程笔者不再详细说明，如果有读者仍有疑问，请参照 2.2.2 小节。相信读者们应该能理解我们做该配置的原因了，因为系统要使用 Android SDK 的一些工具，如 adb.exe 等。接下来验证一下，在命令行中输入命令：adb-h，如果出现如下信息则表示配置成功，如图 2.23 所示。

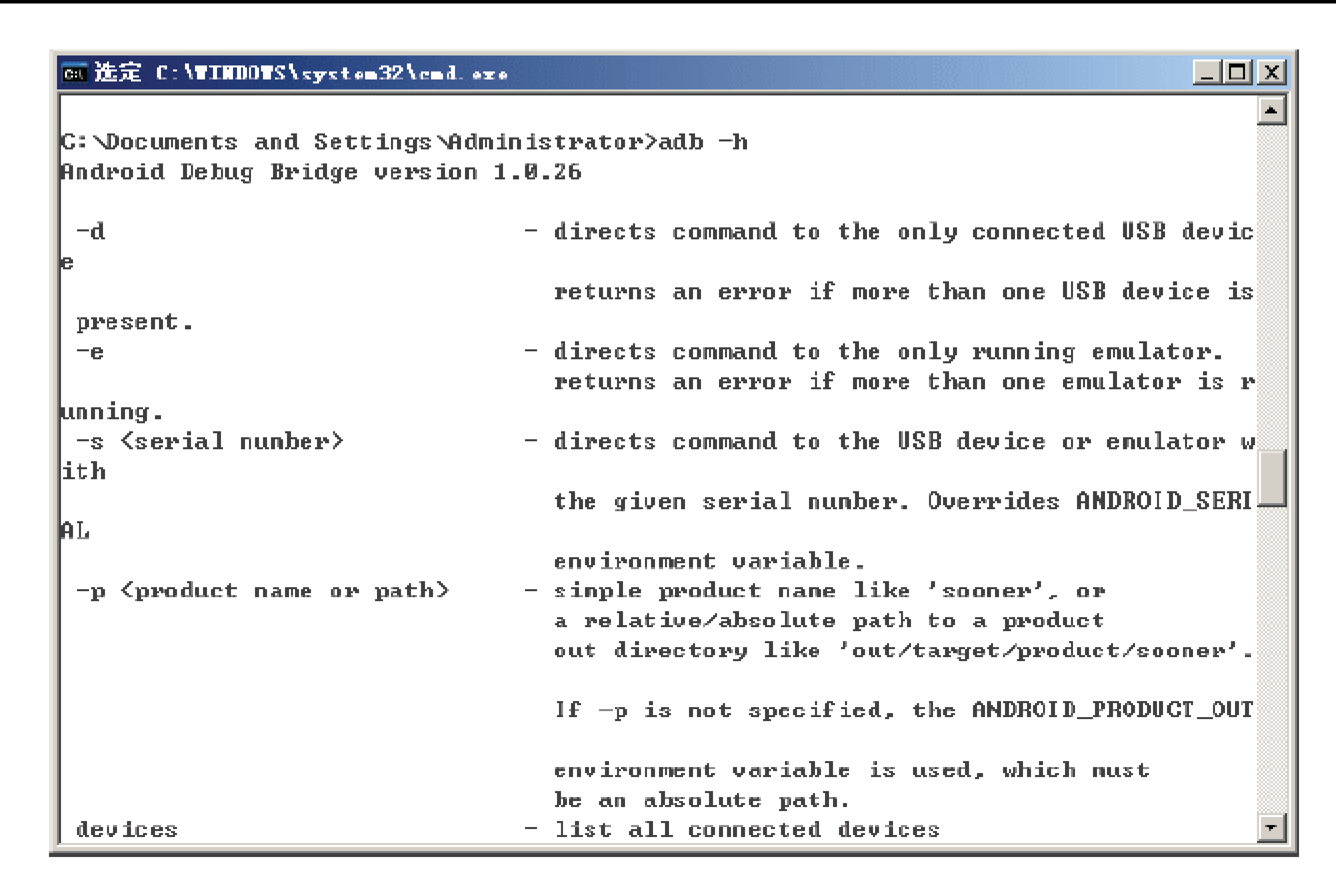


图 2.23　测试 Android SDK 安装是否成功

2.5　下载 ADT

现在我们安装的 JDK 与 Android SDK“各自为政”，还没有很好地结合起来，这对于我们 Android 程序的开发无疑是很不方便的。这个问题可以通过 Android 开发工具（ADT）顺利解决，它可以帮助我们将两者紧密地连接起来。

2.5.1　下载 ADT

首先运行 Eclipse，接着选择 help 菜单中的 Insatall NewSoftware 选项。弹出 Available Software 对话框，如图 2.24 所示。在 Work with 编辑框中输入 https://dl-ssl.google.com/android/eclipse/。

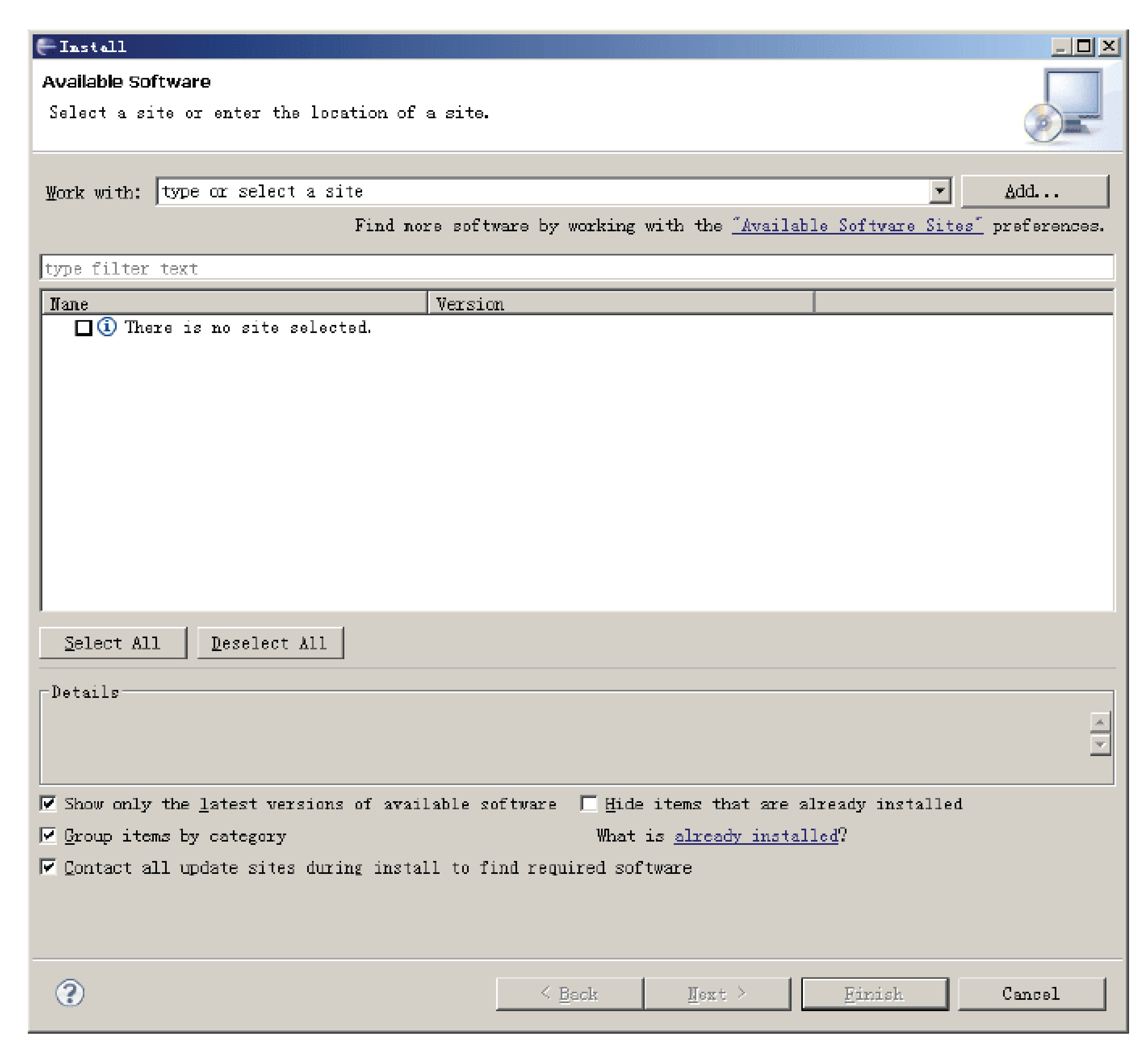


图 2.24　Available Software 对话框

输入后单击 Add 按钮，会出现如图 2.25 所示的信息，将提示要下载的工具全部勾选，然后进入下一步，接下来同意证书，完成下载。完成后重新启动 Eclipse。如果不能正常下载可以尝试输入 http://dl-ssl.google.com/android/eclipse/站点下载。

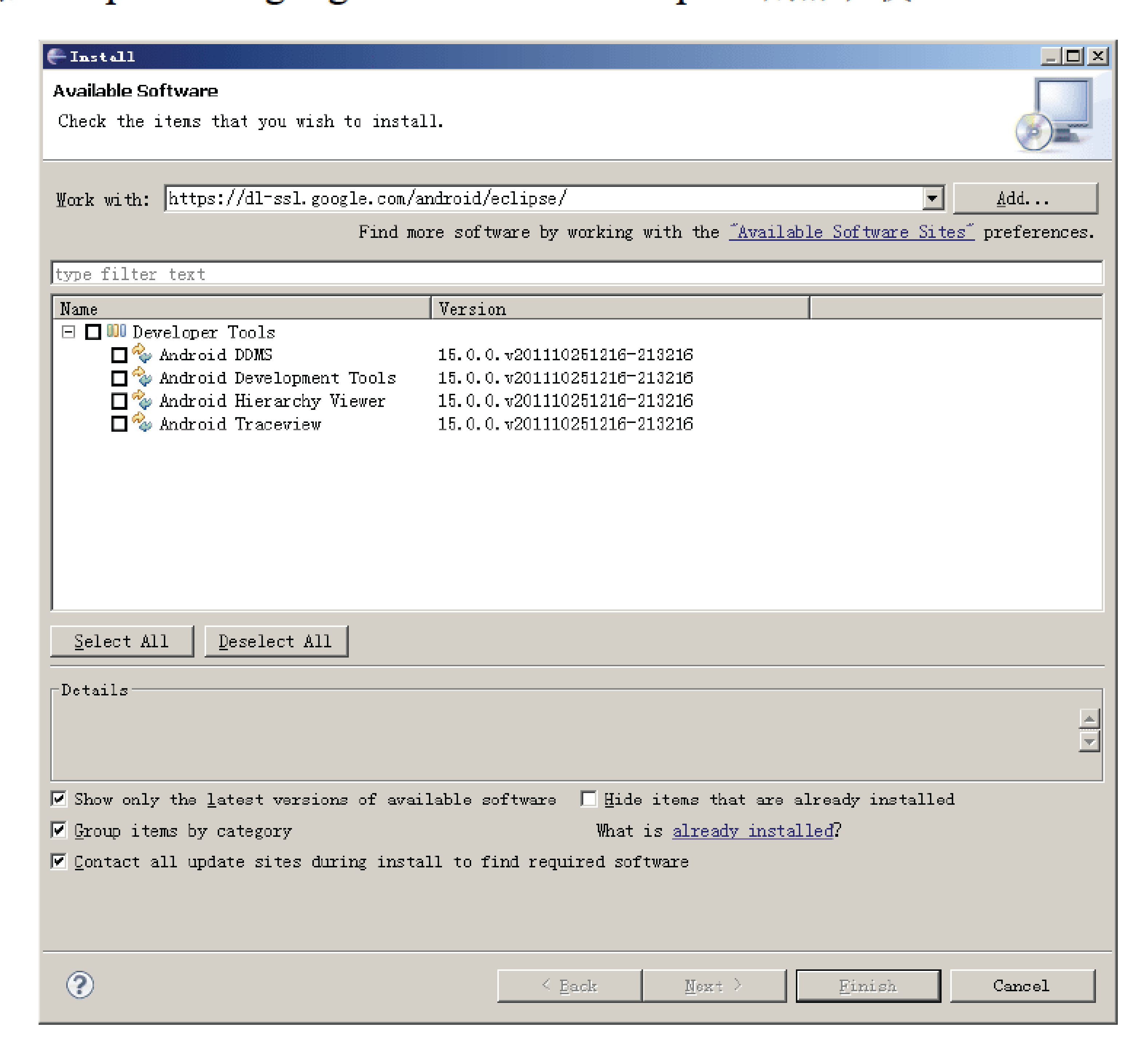


图 2.25　选择开发工具

2.5.2　为 Eclipse 设置 SDK 路径

重启 Eclipse 后你会在 Windows 菜单下的 Preferences 选项中看到 Android 项，如图 2.26 所示。

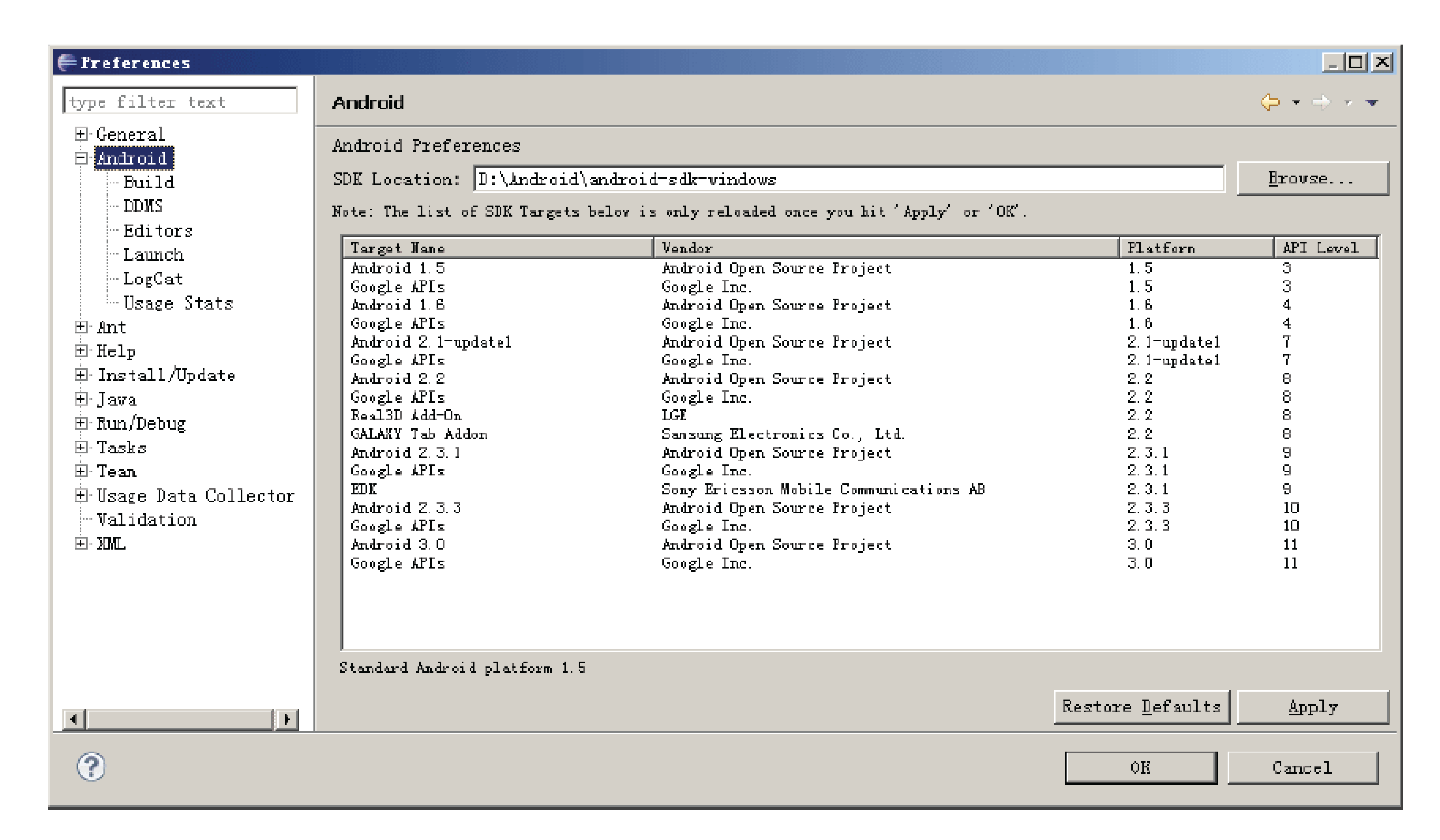


图 2.26　Android 选项卡

在 SDK Location 框中输入 Android SDK 的路径，这样 Eclipse 就可以使用 Android SDK 了。

2.6　新建模拟器

将一切都设置完成后，我们就可以在 PC 机上运行模拟器了。当然我们也可以通过真机进行开发，毕竟很多功能是模拟器无法具备的，比如录音功能、摄像功能等。但是模拟器同样有模拟器的好处，比如它可以很方便地创建出多个对象。

2.6.1　新建 AVD

打开 Eclipse 的 Windows 菜单，选中 Android SDK and AVD Manager 选项，出现如图 2.27 所示的对话框。

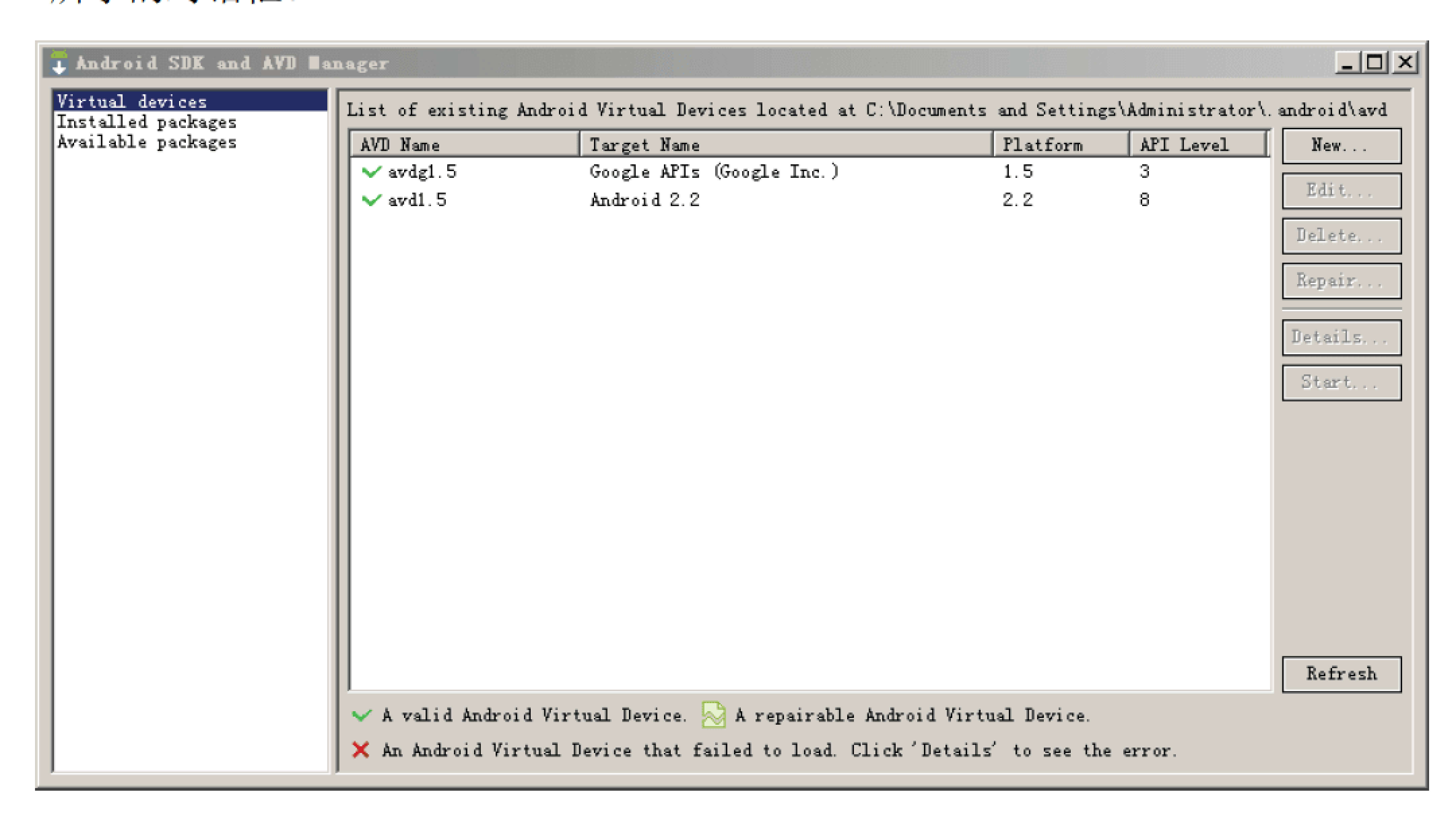


图 2.27　Android SDK and AVD Manager 对话框

单击 New 按钮，出现如图 2.28 所示模拟器的参数设置框。

AVD 的参数说明如下：

- Name：模拟器的名字。
- Target：Android SDK 版本。
- SD Card：虚拟的 SD 卡大小。
- Snapshot：快照，选中后可以快速启动模拟器，否则要等待很长的一段时间。当然，这也是有代价的，代价是每次关闭的时候需要一段时间保存当前的状态。
- Skin：皮肤，可以选择 G1、3G、Hero 等设备，也可以自己设定屏幕大小，这里选择的是 HVGA。
- Hardware：硬件，一般不需设置，当然也可以通过单击 New 按钮来添加，可供选择的有 GPS support、keyboard support 等。

最后单击 Create AVD 按钮完成创建。

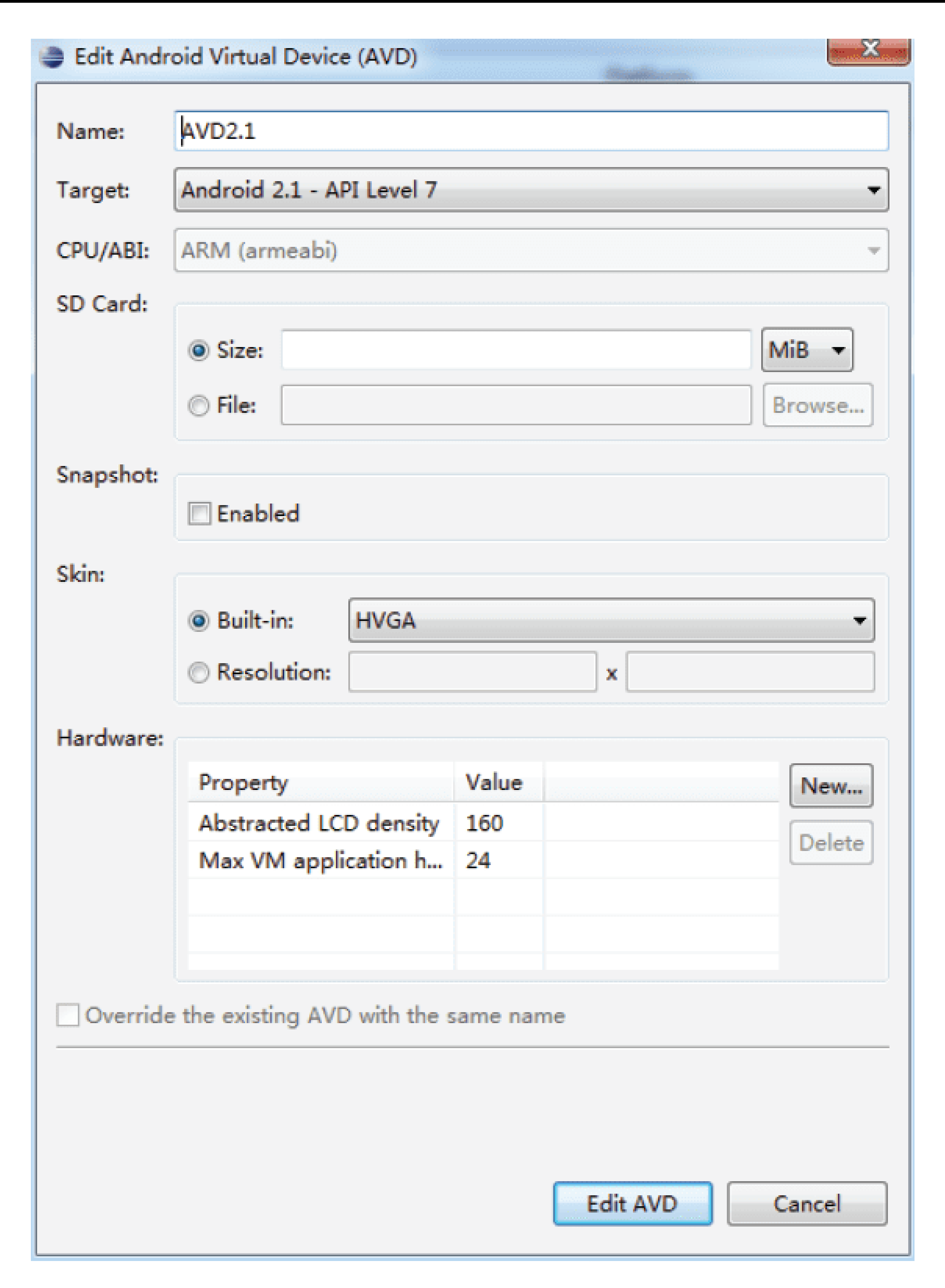


图 2.28　设置模拟器参数

2.6.2　运行模拟器

在 Android SDK and AVD Manager 对话框中选中刚才新建的 AVD，单击 Start 按钮，出现如图 2.29 所示的对话框。

单击 Launch 按钮，启动模拟器，运行效果图如图 2.30 所示。

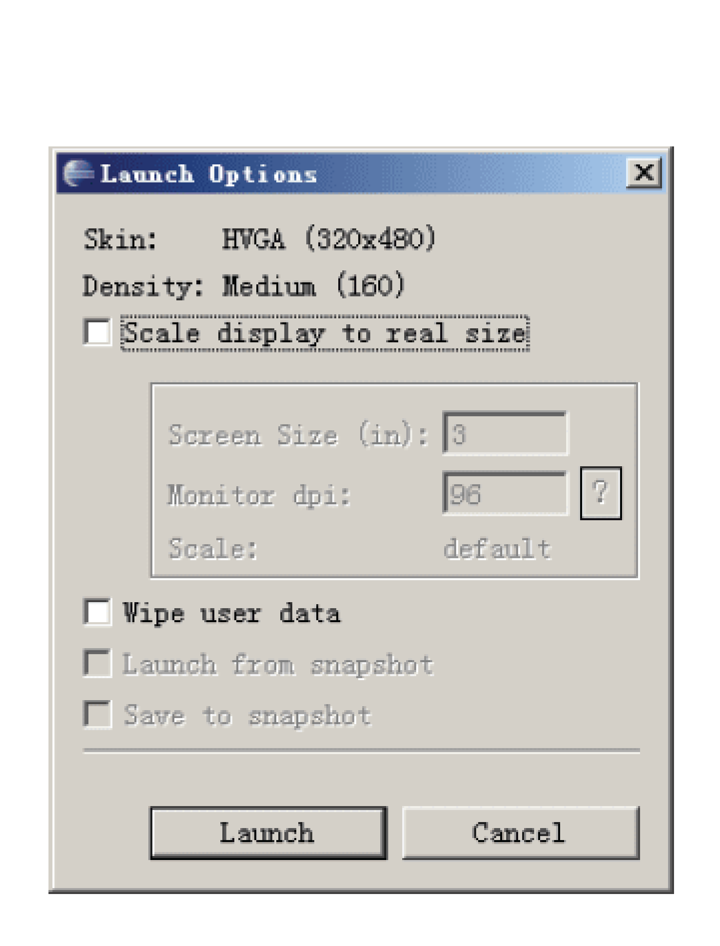


图 2.29　加载选项

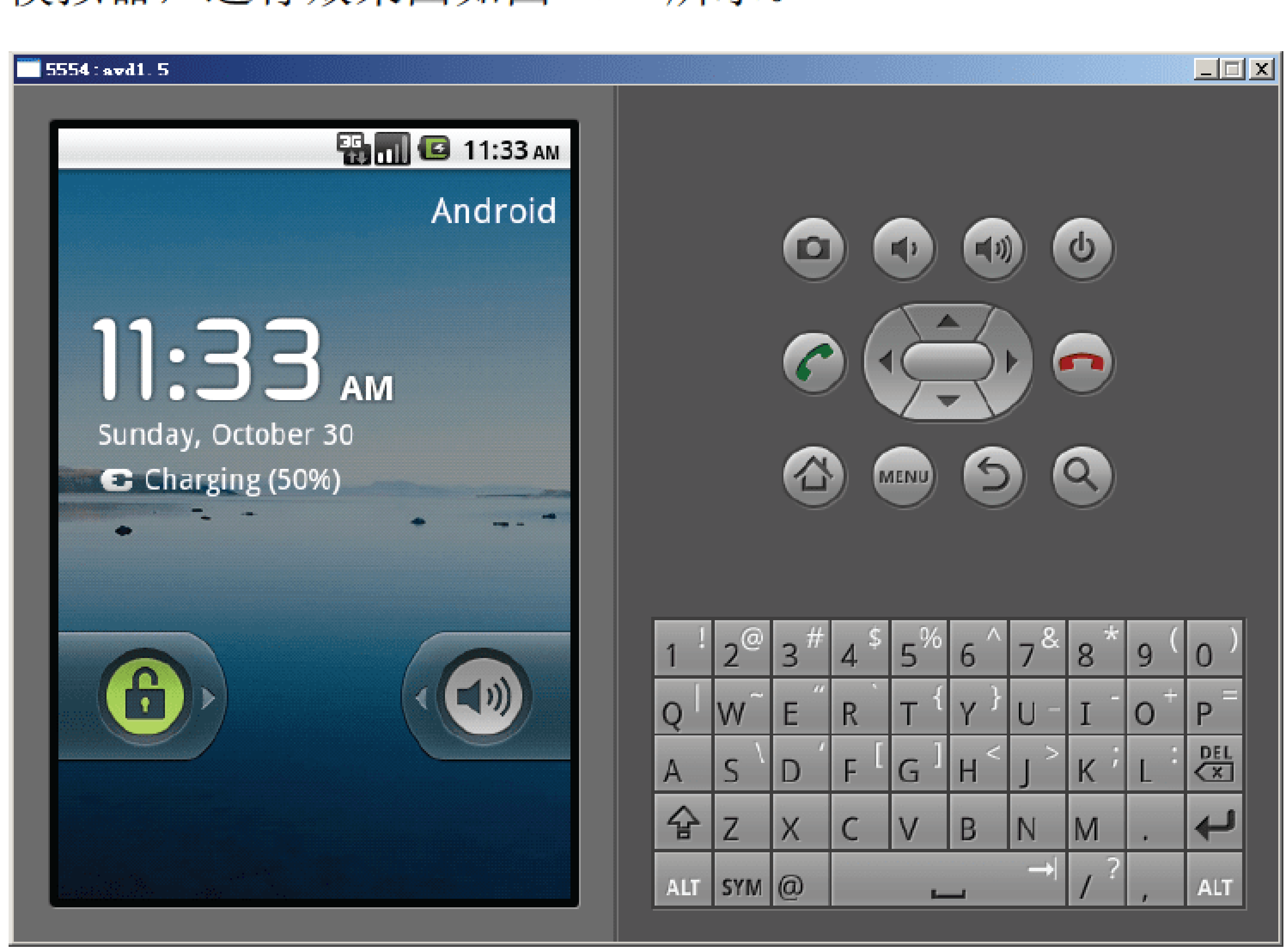


图 2.30　模拟器运行界面

接下来，开始享受 Android 模拟器吧，如果你还没有一台 Android 真机，那么通过它先热热手。如果你已经有了一台真机，那么观察一下模拟器和真机的区别吧。

2.7 真机测试

在开发过程中，我们经常要使用真机帮助我们进行开发，原因有很多，首先模拟器有它的硬伤——很多硬件设备无法支持，如摄像头、重力感应等等；其次，模拟器很消耗 PC 机的系统资源，长时间运行会导致系统运行缓慢。

2.7.1 安装手机驱动

在 Android 2.3 之前，真机的 USB 驱动并无统一标准，所以我们必须自己找到与手机型号对应的 USB 驱动程序。这里使用的开发机型为 HTC Desire。手机驱动的安装这里就不再赘述，因为各个机型都不一样，读者朋友们可以自行学习安装。在安装完成之后，可以将手机与 PC 机连接。这个时候对于 PC 机来说，真机与模拟机已经没有什么分别了。

2.7.2 设置手机

安装好手机驱动后，我们还需对手机进行一些设置，这样才能顺利地用于开发调试。打开系统的菜单，如图 2.31 所示。

选择“设置”按钮，进入如图 2.32 所示的界面。



图 2.31　系统菜单项



图 2.32　设置界面

选择“应用程序”，进入图 2.33 所示的界面，接着选择“开发”选项，进入图 2.34 所示的界面。




图 2.33　应用程序界面



图 2.34　开发界面

将“USB 调试”保持唤醒状态，“允许模仿位置”全部勾选。这样就可以进行真机调试了。在命令行输入：

```
adb devices
```

如果出现你的手机信息，则表示安装成功，提示信息如图 2.35 所示。

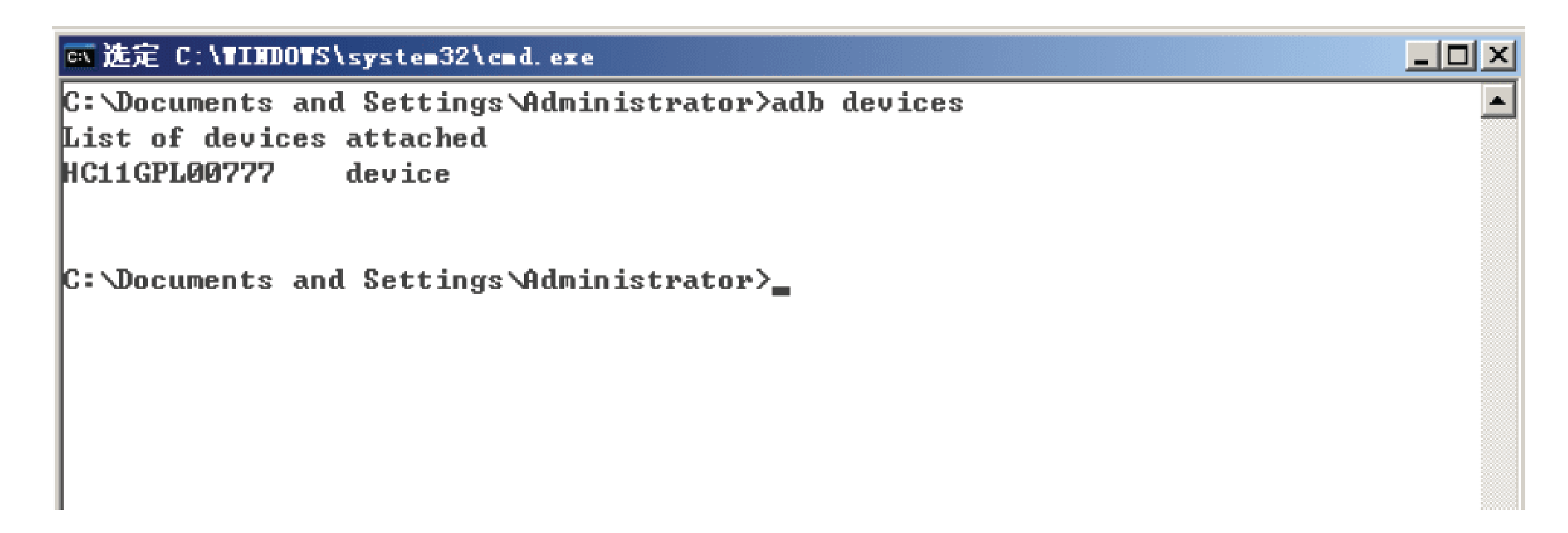


图 2.35　查看连接设备

2.8　小　　结

本章讲解了 Windows XP 下的 Android 开发环境的搭建。从网上下载各种安装包开始，到 JDK、Eclipse、Android SDK 以及 ADT 的安装和理解，最后讲解了模拟器和真机调试的方法。本章重点讲解了 JDK 以及 Android SDK 的安装，难点是系统环境变量的设置以及设置的目的和意义。下一章我们将创建第一个应用程序——HelloWorld。

第 3 章　创建第一个程序——HelloWorld

通过第 2 章的讲解，相信大家已经在计算机上拥有一个可以开发 Android 应用程序的环境了。接下来，我们就开始编写第一个 Android 程序——HelloWorld！HelloWorld 作为最简单也最经典的应用，我们希望通过它，使读者学会如何新建 Android 程序、阅读程序以及调试和运行程序。

3.1　新建第一个程序

从本节开始我们进入具体的代码编写阶段，愉快的 Android 学习之旅就此展开。接下来开始新建第一个工程，如果你对 Eclipse 还不是那么熟悉，那么在本章你可以对它有一个初步的认识，学会一些基本的使用方法。

3.1.1　新建工程

首先打开 Eclipse，在 File 菜单中单击 New|Project 命令，如图 3.1 所示。

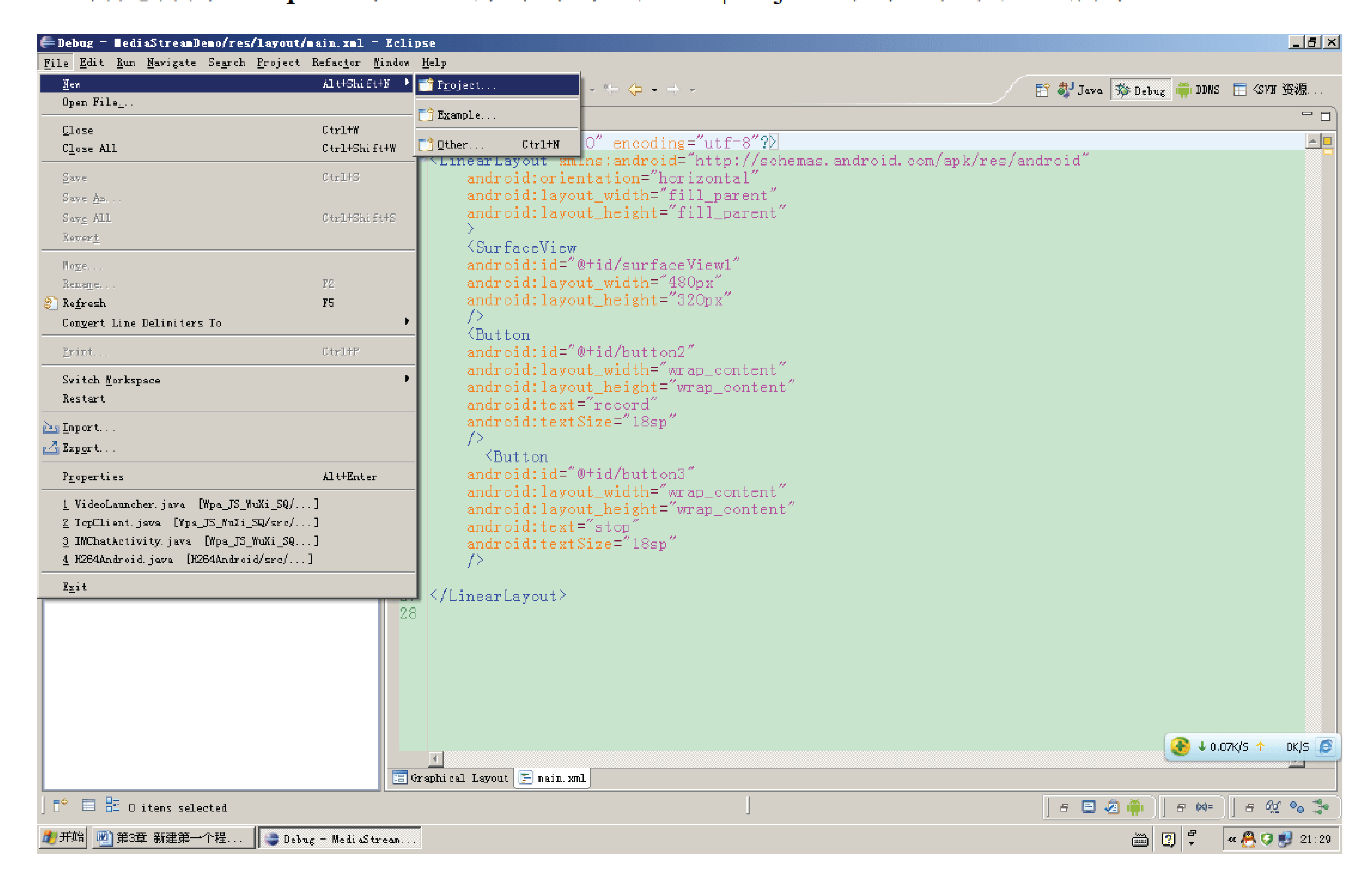


图 3.1　新建工程入口

当然也可以选择在工程浏览器中单击右键，在弹出的菜单中选择 New Project，同样可以新建工程，如图 3.2 所示。

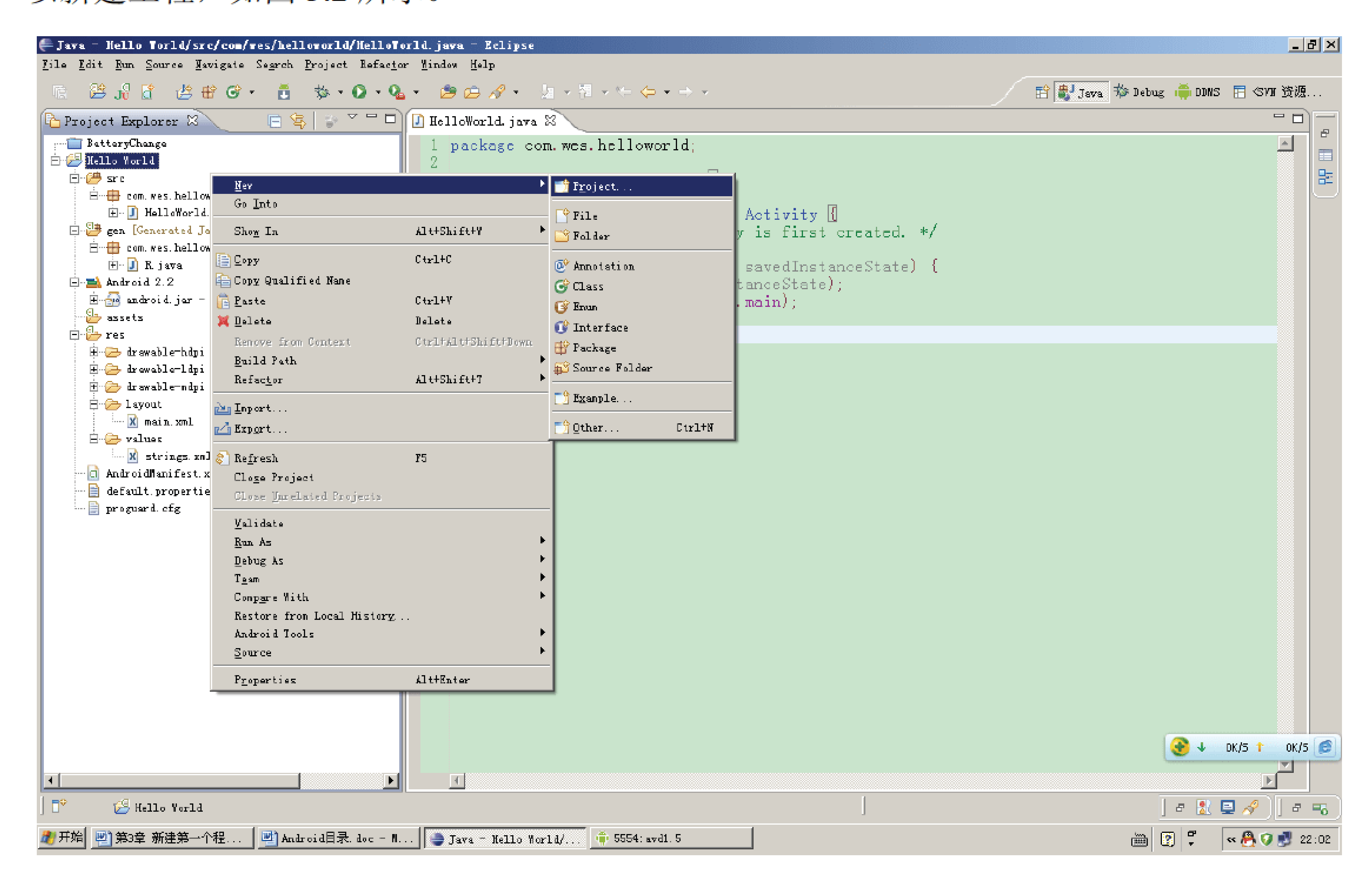


图 3.2　通过工程管理器新建工程

接着，在弹出的对话框中选择 Android，再选择 Android Project 选项，如图 3.3 所示。

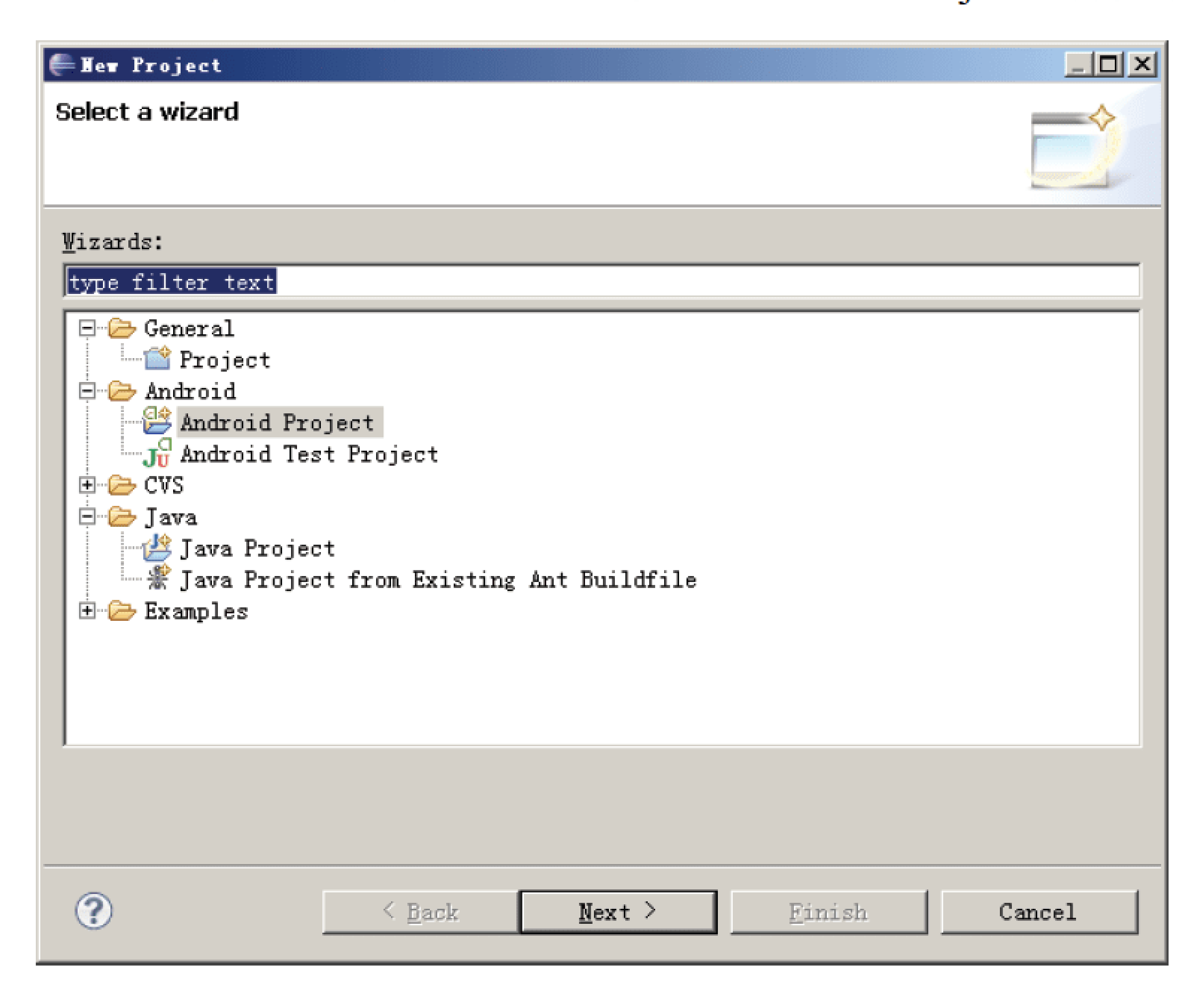


图 3.3　新建工程

单击 Next 按钮后，弹出如图 3.4 所示对话框，在该对话框中需要填写各类信息，包括：

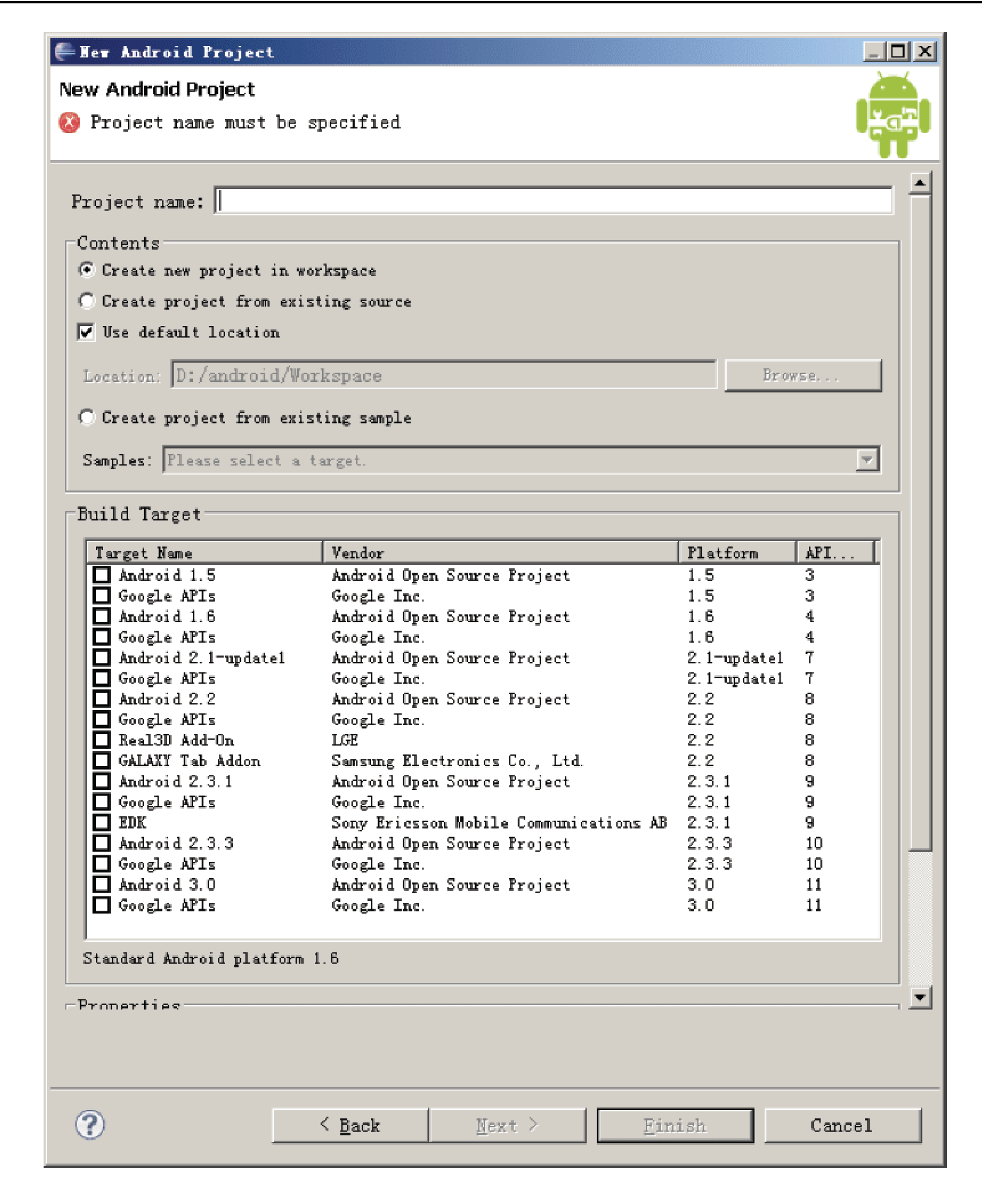


图 3.4　新建 Android 工程

（1）Project Name：工程名，比如 HelloWorld。

（2）Content：勾选 Create new project（在工作空间中新建工程）和 Use default location（使用默认工作空间）两项。工作空间大家还记得么？就是你工程文件存储的地方啦。

（3）Build Target：选择 Android SDK 的版本，在你希望创建的版本前打钩就可以了。

（4）Properties：属性，包括应用名（Application Name）、包名（Package Name）、活动名（Create Activity）和最低 SDK 版本（Min SDK Version）。

填写完毕后单击 Finish 按钮，这样一个新的 Android 工程就新建完成了。

3.1.2　运行程序

新建完成 Android 程序后，我们接下来做什么，开始编写代码了么？不，HelloWorld 不需要编写任何代码就可以运行，接下来就运行我们新建的第一个程序。

在工程浏览器中选择新建好的 HelloWorld 工程，单击右键在菜单中选择 Run as，在弹出的菜单中选择 Android Application，如图 3.5 所示。

单击 Android Application 后就可以进入模拟机的启动画面了。等待一段时间，模拟器启动完毕后，我们就可以看到程序的运行效果了，如图 3.6 所示。

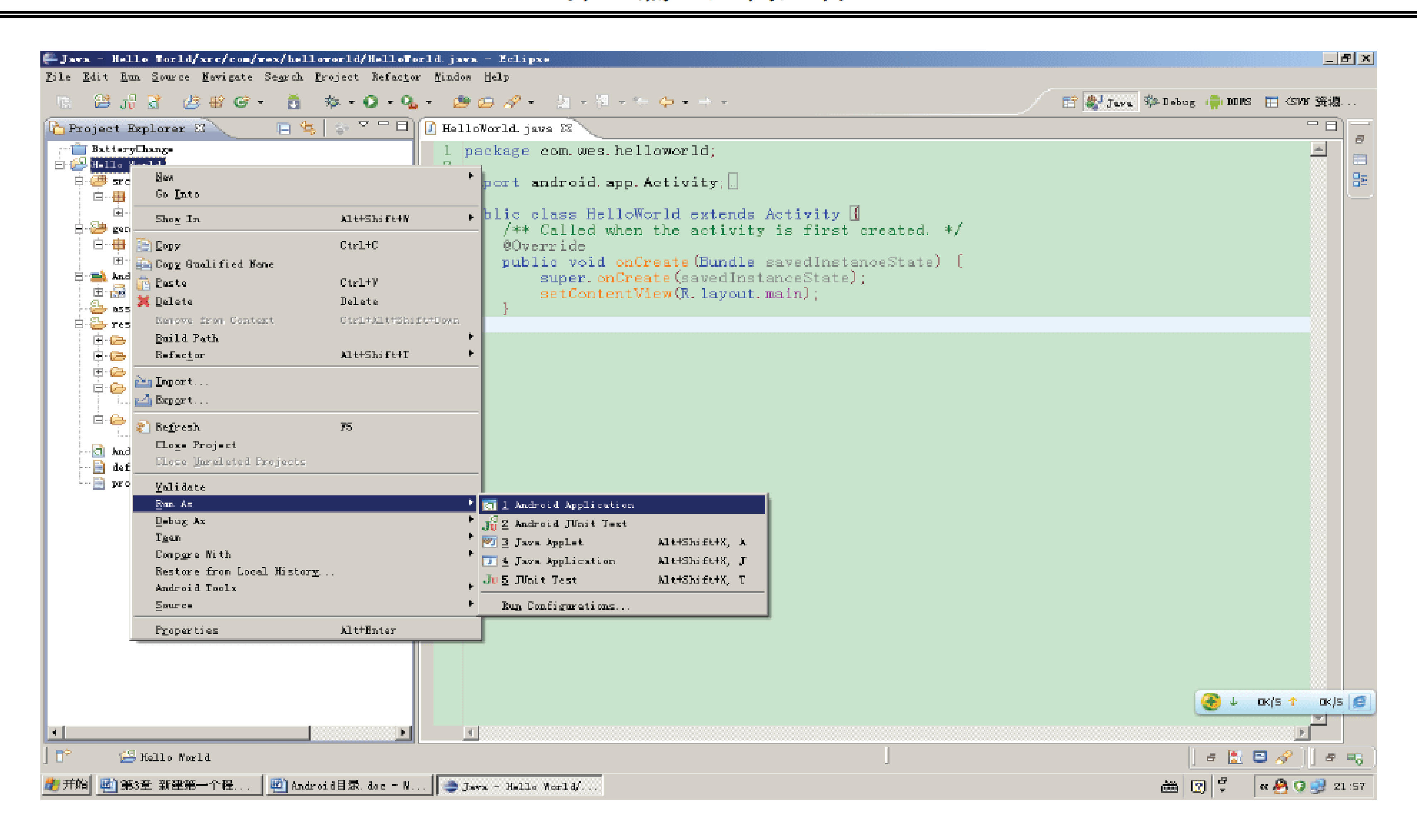


图 3.5　运行程序

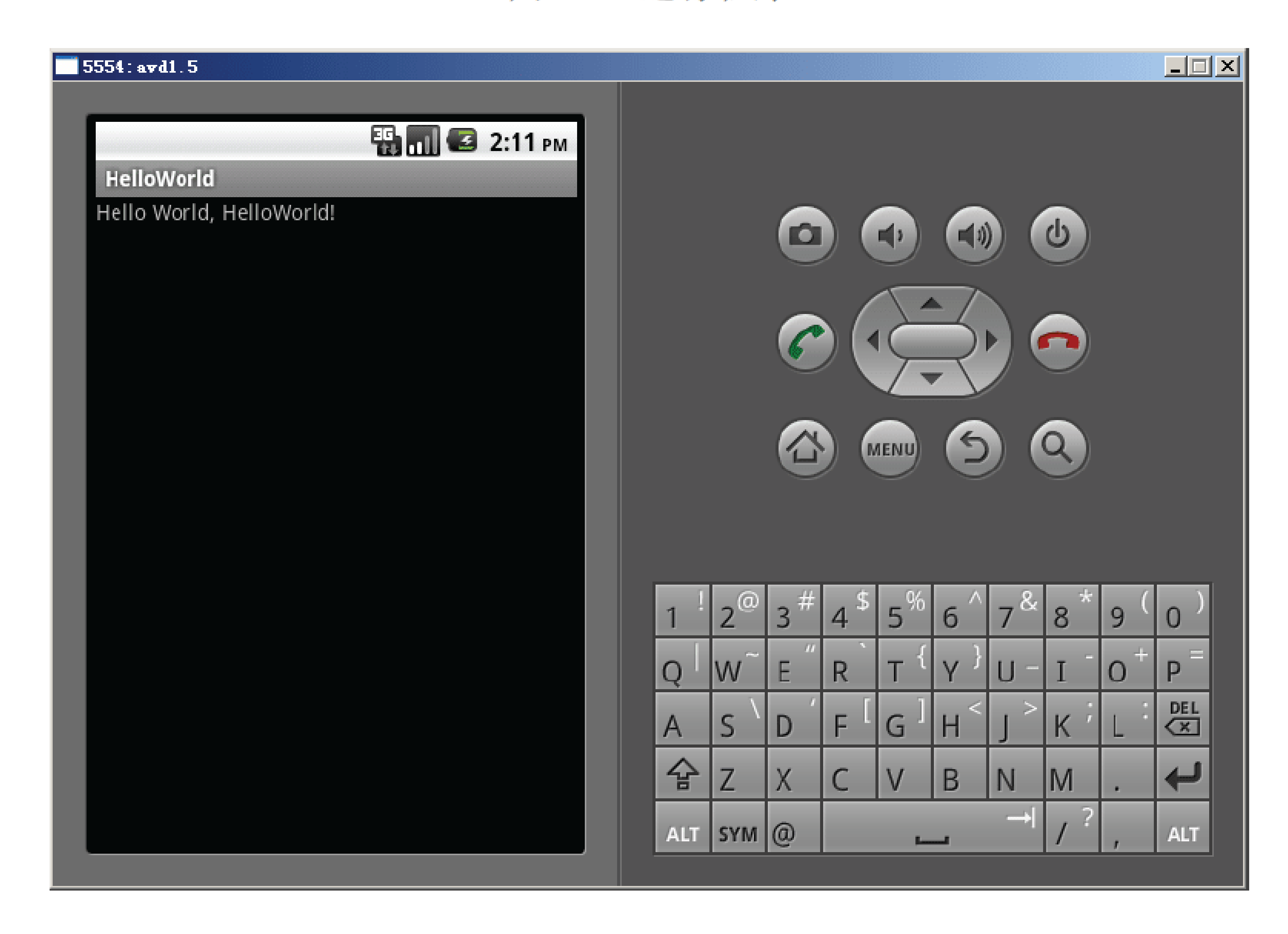


图 3.6　运行效果

3.2　认识 HelloWorld

在 3.1 节我们成功运行了新建的 HelloWorld 程序，但是程序到底是怎么运行的呢？我们的代码又是怎样编写的呢？本节就要一步步揭开 Android 应用程序编写的神秘面纱，从整体上认识 Android 工程结构、各个文件夹的作用以及一些基础的代码意义。

3.2.1　首识 Android 工程

现在我们回过头来再次认识 HelloWorld，并了解工程的各个组成部分的作用。在工程

浏览器中打开 HelloWorld 的层级结构，如图 3.7 所示。

首先我们看到了 src 文件夹，该文件夹包含的为源代码，可以说是工程中最主要的部分，双击 HelloWorld.java 文件，我们就可以看到其中的代码了，如图 3.8 所示。

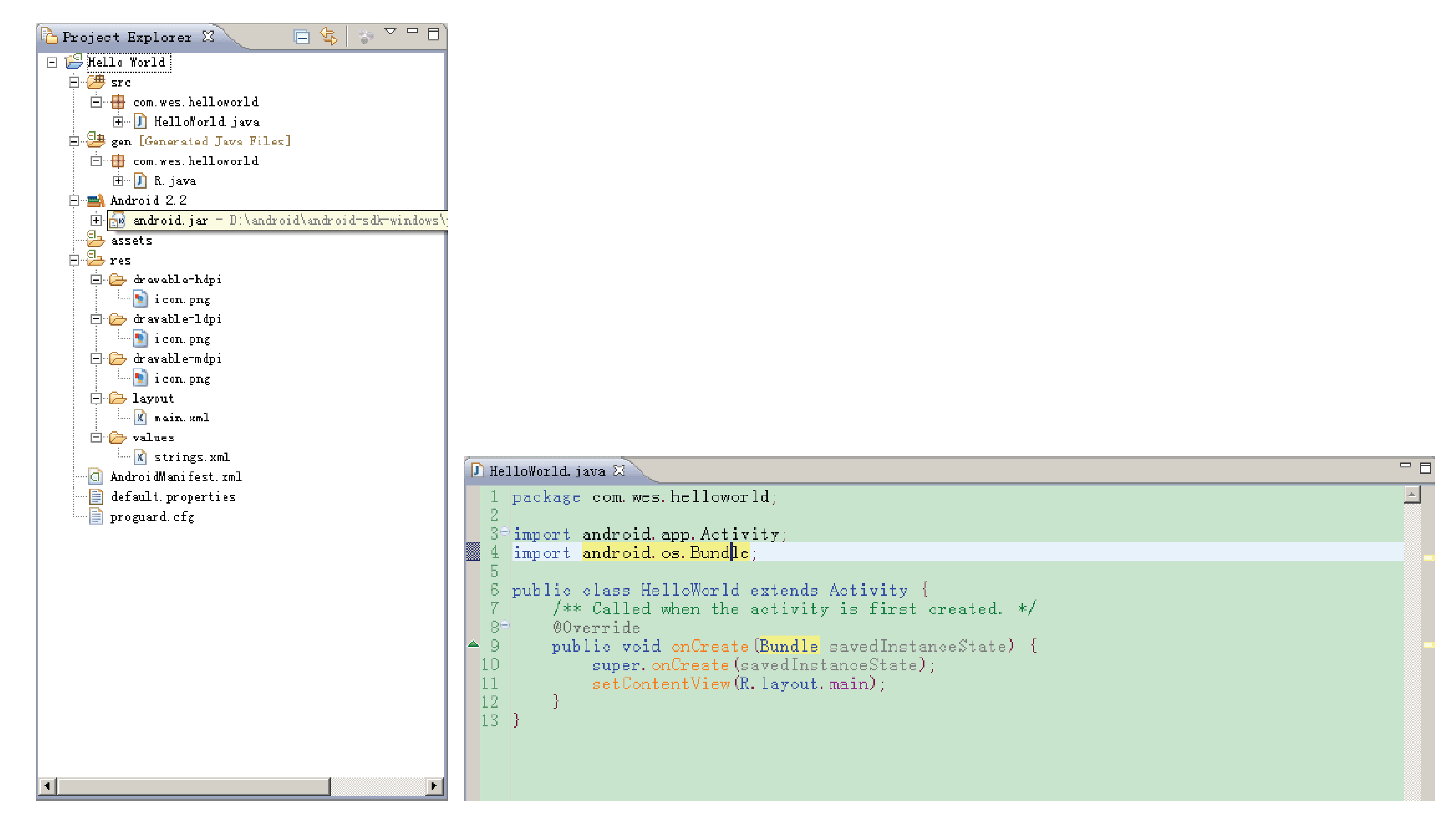


图 3.7 工程目录　　　　图 3.8 源代码

从图 3.8 中，我们知道该示例程序总共也只有 13 行代码，去掉其中的空行、导入声明和花括号，真正的代码只有 4 行！那么接下来我们就一起来分析分析，这些代码到底有什么作用，他们是怎么控制模拟机显示如图 3.6 的运行效果的呢？

首先，第一行的意义相信有 Java 基础的读者都知道作用，声明该类属于 com.wes.helloworld 包。

第 3、4 行，声明导入的类，分别导入了 android.app.Activity 类和 android.os.Bundle 类，这两个类的作用又是什么呢？Activity 类是所有的 Android 应用必须要继承的一个类，它可以被称为一个活动，从某种意义上说，Android 所有的应用都是活动。Bundle 类是捆绑的意思，用来保存一些重要的数据。在以后的编程中，读者朋友们会经常与这两个类打交道，这里只做一个简单的介绍，本书之后的章节会有详细讲解。

第 6 行，声明 public class 并声明其继承自 Activity。

第 9 行，onCreate()方法，这是继承自 Activity 的方法，是 Activity 的生命周期之一，也是 Activity 开始的入口。事实上，Activity 提供了若干方法管理其生命周期，例如：

- ❑ 创建 Activity 时：onCreate()。
- ❑ 开始 Activity 时：onStart()。
- ❑ 暂停 Activity 时：onPause()。
- ❑ 停止 Activity 时：onStop()。
- ❑ 销毁 Activity 时：onDestory()。

当然，这里同样只是举了几个最常用的生命周期，还有更多更详细的知识，我们会在之后的章节中再进行深入学习。

第 10 行，调用父类的 onCreate()方法，以创建 Activity。

第 11 行，设置界面，方法是将页面与资源文件绑定。setContentView()方法的参数 R.layout.main 就是 R 文件中的布局文件 main.xml 的引用。

3.2.2　认识布局文件

在 3.2.1 小节的末尾我们提到了两个概念，一是布局文件 main.xml，二是 R 文件。本小节就首先讲解布局文件的相关知识。

布局文件位于 res/layout/目录下，具体到 main.xml，它的路径就是 res/layout/main.xml。

找到 main.xml 文件，双击打开它，我们就可以看到布局文件的真面目了，如图 3.9 所示。

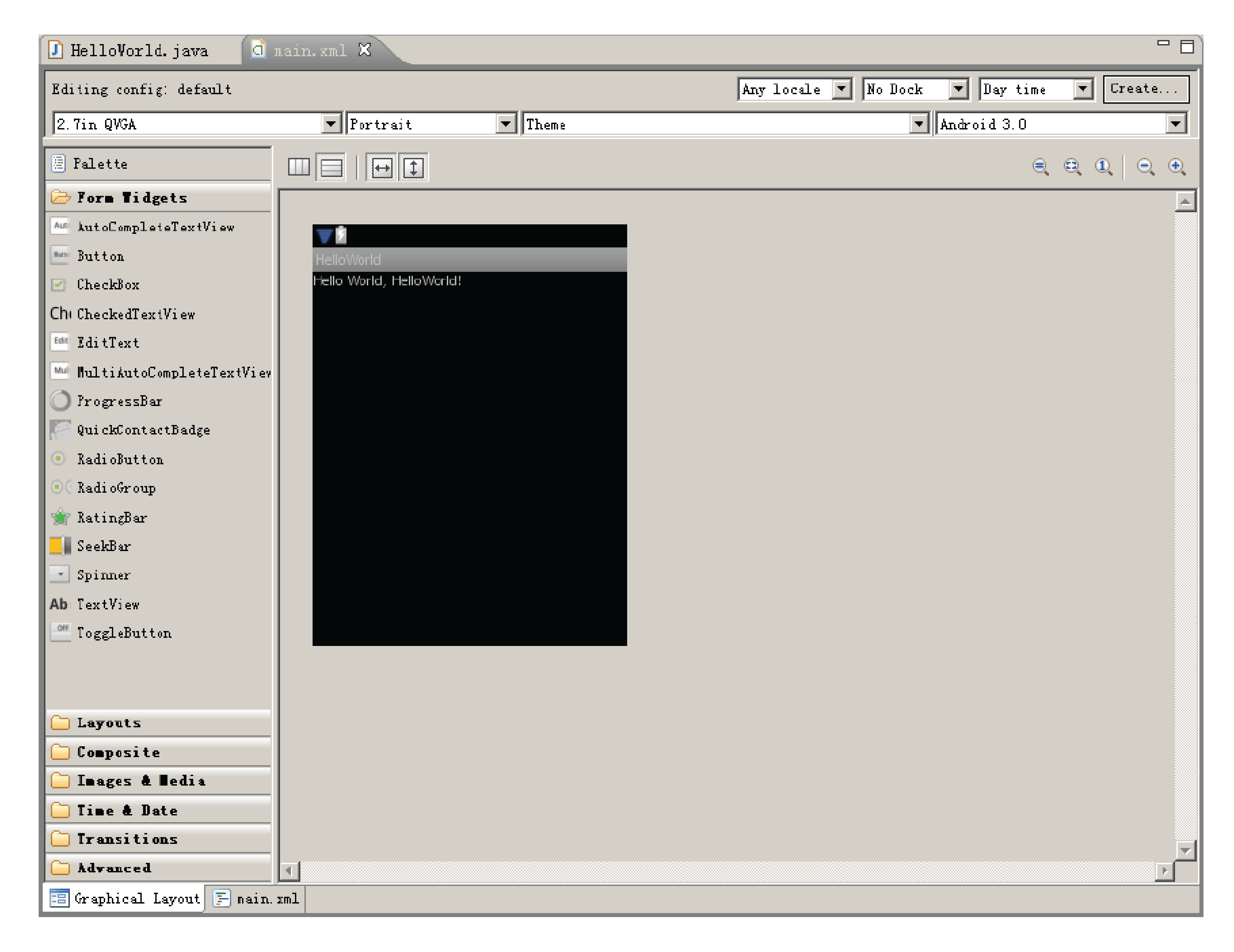


图 3.9　布局文件预览

当然我们看到的只是预览界面，右边的黑色部分就是模拟器的显示屏了，我们可以看到显示屏上已经出现了 HelloWorld 字样了。当然，很多读者会问，代码呢？布局文件的代码在哪里？不要着急，单击底部的 main.xml 标签。这个时候就可以看到具体的代码了，如图 3.10 所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

图 3.10　布局文件代码

由图 3.10，我们看到 main.xml 文件总共有 12 行代码，接着我们就简单认识一下布局文件的简单代码。

第 1 行：xml 文件头，每个 xml 文件必须的声明，其中包括版本和编码方式。

第 2～6 行：线性布局的开始节点，其中包括了若干属性，如方向、宽度、高度等。

第 7～11 行：完整的文本视图节点，其中包括的属性有宽度、高度和现实内容。

第 12 行：线性布局的结束节点。

经过上文的分析，我们知道这 12 行代码要表达的意思是在一个线性布局中插入一个文本视图，在文本视图中显示文字内容，内容为“@string/hello”。

3.2.3　认识值文件

上小节中我们知道文本视图 TextView 中的属性：

```
Android: text ="@string/hello"
```

表示在文本视图中显示内容“@string/hello”。可是事实上，我们在模拟器的显示屏中看到的却是 HelloWorld 字样啊，这又是怎么回事呢？

事实上，这里的“@string/hello”只是表示要显示的内容的引用，真正的内容还藏在其他的地方呢！@标志表示 xml 之后的是需要解析的内容而非要显示的内容，如果去掉了@符号，那么模拟器的显示屏上就真的显示“string/hello”字样了，有兴趣的朋友可以试验一下。

言归正传，既然@标志的意思是之后的内容需要解析，那么怎么解析又是一个新的问题。“@string/hello”的实际内容是：解析到 string 类型的节点名为 hello 的内容。

也许这么说读者还不是很明白，为了使读者更清晰，我们就来看一看这 string.xml 文件又是何方神圣。在/res/values 文件目录中找到 string.xml 文件，双击打开，显示如图 3.11 所示。

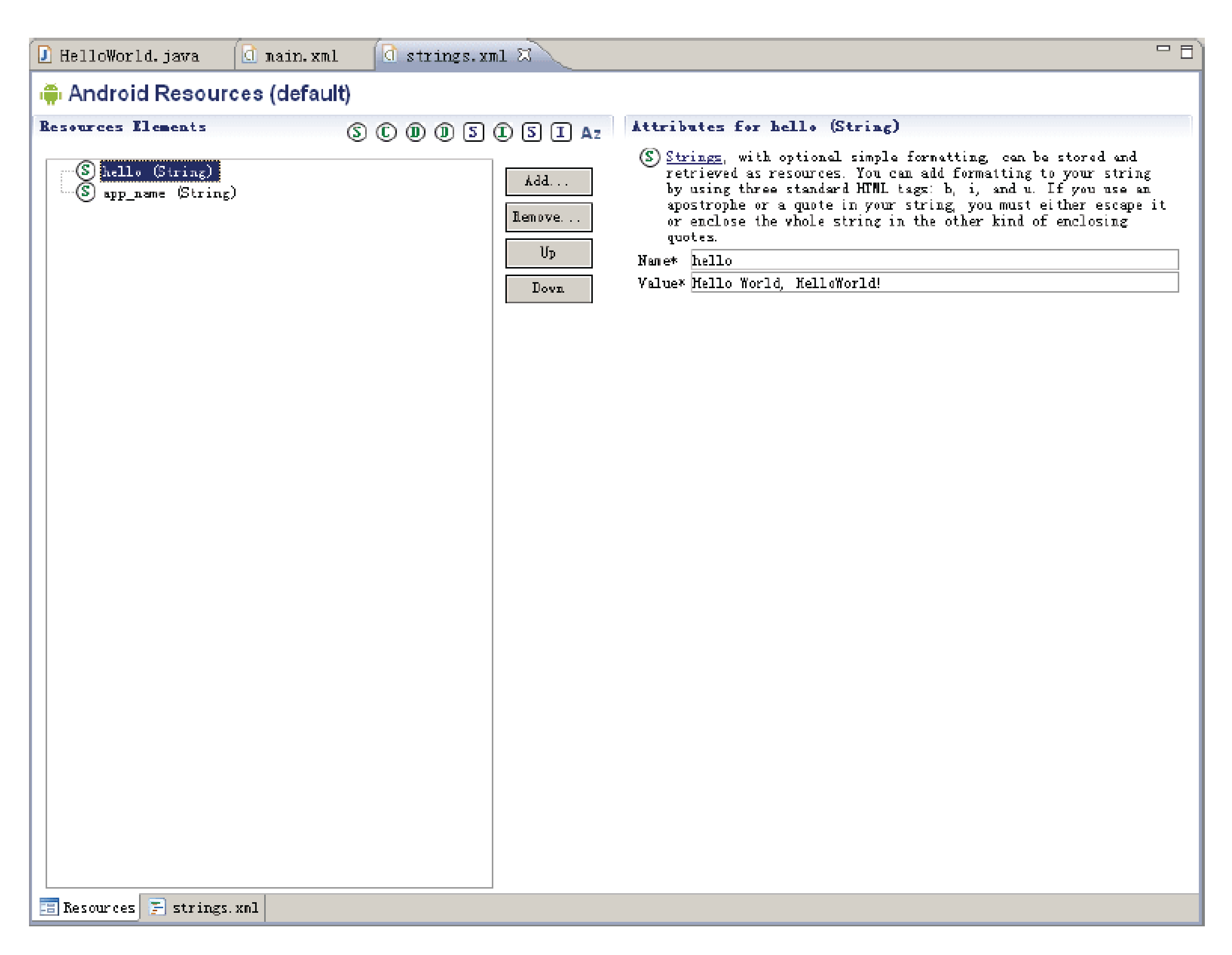


图 3.11　string.xml 文件预览

选中左侧的 hello（String）选项，在右侧我们可以看到 Attributes for hello（String）框，其中就可以看到两个编辑框，一个是 Name，其中显示 hello，另一个是 Value，其中显示 Hello World，HelloWorld。

看到这里，读者朋友们应该明白了，这里的 Value 才是我们真正要显示的内容啊，hello 只是一个名字，用来给 xml 解析。就好像是你的朋友喊你的名字，名字只是一个代号，你朋友真正要找的是你这个人而不是你的名字。

单击底部的 strings.xml 文件，我们可以看到真正的 strings.xml 的代码，如图 3.12 所示。

```
HelloWorld.java | main.xml | strings.xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="hello">Hello World, HelloWorld!</string>
4     <string name="app_name">HelloWorld</string>
5 </resources>
6 
```

图 3.12　string.xml 代码

- ❑ 第 1 行依然是 xml 文件头，不再赘述。
- ❑ 第 2 行与第 5 行：resources 的完整节点。
- ❑ 第 3 行：string 类型节点，其名字是“hello”，值是 Hello World，HelloWorld！
- ❑ 第 4 行：string 类型节点，其名字是“app_name”，值是 HelloWorld。

3.2.4　认识 R 文件

我们已经知道布局文件通过@符号与值文件中的值连接起来，而布局文件是通过源代码中的 setContentView（R.layout.xx）方法绑定到一起。其中的 R.layout.xx 就起到了@的作用。该参数的意义是：通过 R 文件找到 layout 文件中的 xx 布局文件。例如，要找到 main.xml 布局文件，其参数就是 R.layout.main。

那么 R 文件在哪里？R 文件又是怎么找到 main.xml 文件的呢?接下来，我们就来探究 R 文件。

R 文件位于/ge/<package name>/R.java 目录下，它就好比是一个联系薄，记录着所有可使用资源的 Id，通过这些 Id，我们就可以很方便地在程序中使用这些资源了。双击 R.java 文件，我们来看看这个 Android 应用的“神经中枢”的“庐山真面目”，如图 3.13 所示。

```
HelloWorld.java | main.xml | strings.xml | R.java
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.wes.helloworld;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

图 3.13　R 文件代码

第 1～6 行是由系统自动生成的注释，它的意思是：这个类是由 aapt 工具通过它找到的资源数据自动生成的，它不能被手动修改。aapt 也就是 Android Asset Packaging Tool，即 Android 资源打包工具。该工具一般由 Eclipse 调用，我们不需要主动去使用。

第 8～23 行就是 aapt 自动生成的代码了，该类的名字就是 R，其下定义了一些常量，包括：

- Drawable：图片资源，目前只包含 icon，其 Id 为 0×7f020000。
- Layout：布局文件资源，目前只包含 main.xml，其 Id 为 0×f030000。
- String：string 类型资源，目前包含 app_name 以及 hello。hello 我们已经使用了，而 app_name 就是该应用的名字。

更多的关于 R 文件的知识将会在第 6 章使用程序资源中讲解。

3.2.5　认识注册文件

最后，我们还需要认识 Android 工程中另一个重要的文件——AndroidManifest.xml 文件，也就是 Android 注册文件。它直接位于工程目录下，与 src、res 等文件夹平级，由此可以看到它的重要性了。双击打开注册文件，显示如图 3.14 所示。

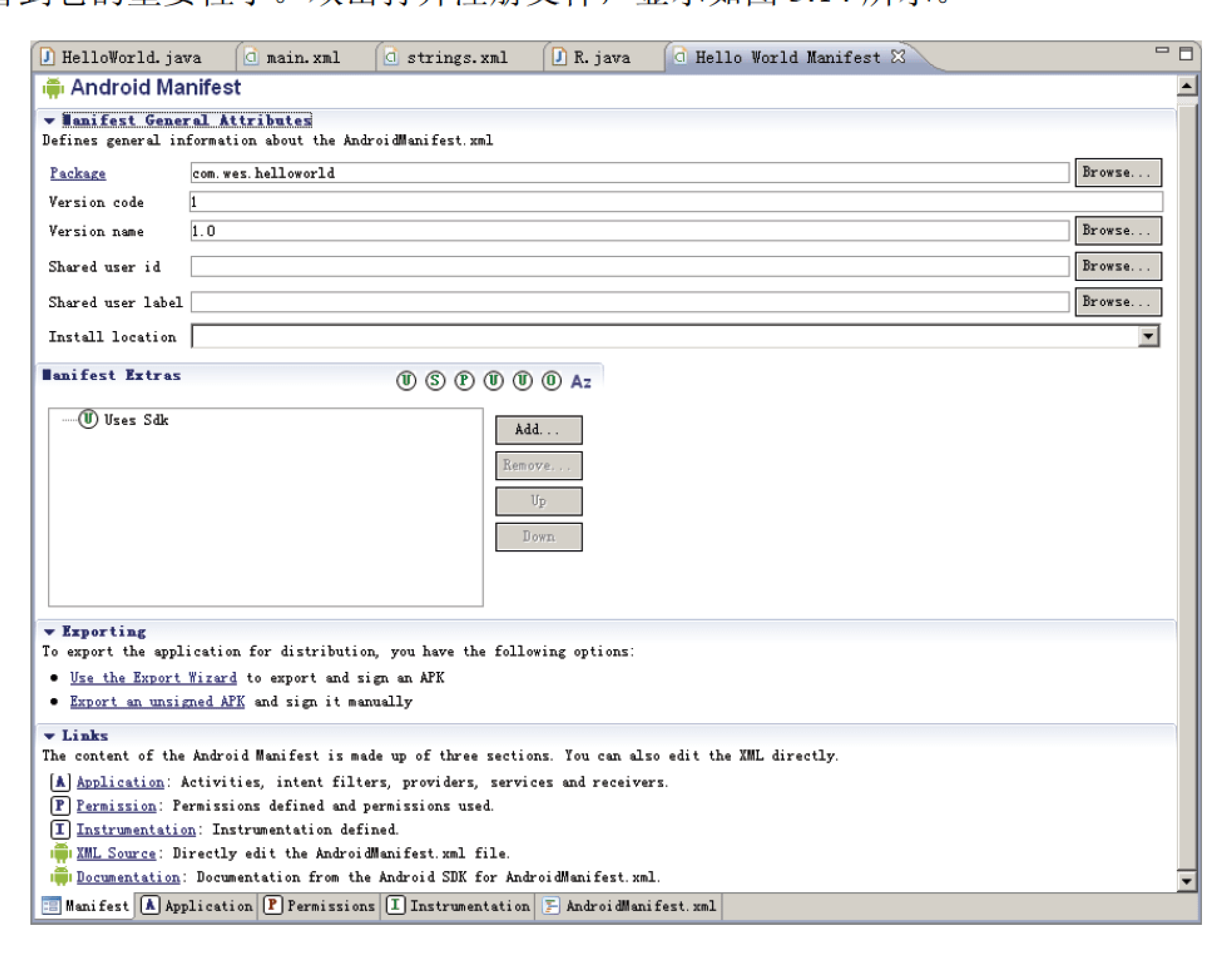


图 3.14　注册文件信息

该界面中显示了一些应用的相关信息，如包名、版本、使用的 SDK 版本等。单击底部最右边的 AndroidManifest.xml 标签可以看到其具体的代码，如图 3.15 所示。

让我们从第 2 行开始阅读该注册文件，并从中读取我们需要的信息。

第 2 行：manifest 节点，其属性包括：

- 包名：注册了 com.wes.helloworld 这个完整包名字。

HelloWorld.java | main.xml | strings.xml | R.java | Hello World Manifest

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3       package="com.wes.helloworld"
4       android:versionCode="1"
5       android:versionName="1.0">
6     <uses-sdk android:minSdkVersion="8" />
7 
8     <application android:icon="@drawable/icon" android:label="@string/app_name">
9         <activity android:name=".HelloWorld"
10                   android:label="@string/app_name">
11             <intent-filter>
12                 <action android:name="android.intent.action.MAIN" />
13                 <category android:name="android.intent.category.LAUNCHER" />
14             </intent-filter>
15         </activity>
16 
17     </application>
18 </manifest>
```

图 3.15　注册文件代码

- 版本号：注册了该应用的版本号，此处版本号为 1。版本号的用处是为了方便以后的应用发布管理，可以有效地避免因程序版本混乱造成无法更新等问题。
- 版本名称：注册了该应用的版本名称，这是显示给用户的。也就是说，当用户从网上下载应用时，版本信息将呈现给用户。

第 6 行：使用的 SDK，指定了最低版本为 8，也就是 2.2 版本。这样指定了以后，程序就可以在不同的 SDK 版本之上运行并移植了。

第 8 行：应用节点，其属性包括：

- 图标：android：icon=“@drawable/icon”，也许聪明的你已经读懂了本句代码的意义，找到 drawable 文件夹下的 icon 图片。如果你希望程序更加生动吸引人，你可以通过修改本句代码来修改应用图标，默认的图标是一个小机器人。不要小看这个小小的图标，如果没有修改它，那么你的程序很可能“泯然众人矣”，消失在茫茫多的应用中，如果能使用一张很炫的图片作为程序的图标则能给人一种这个程序很强大的感觉。
- 标签：指定了应用的名称，同样的 android：label=“@string/app_name”，向用户显示该应用的名称，该名称是在新建工程时填写的，还记得么？就是 ApplicationName 属性。

第 9 行：活动节点，注册了活动的名称和活动的标签。要注意每个 Activity 都必须在 Androidmenifest 文件中注册，否则系统将找不到该活动。其中 name 的属性值是“.HelloWorld”，这是一个缩写，如果填写全称可以加上包名，比如 com.wes.helloworld.HelloWorld。

第 11～14 行：Intent 过滤器。要理解本行代码我们首先要了解——每个 Intent 都是由 Intent 启动的，意图（Intent）的相关知识会在之后的章节中讲解。而 Intent 过滤器就是为 Intent 准备的，现在读者只需要知道：该过滤器指定了该 Activity 是本应用的程序主入口。

到这里呢，最简单的注册文件我们就解读完毕了，当然注册文件还有更多的属性，如 ContentProvider 属性、允许使用权限属性等，这在以后的学习中会穿插进行讲解。

3.3　调 试 程 序

调试程序是在应用开发时必不可少的一个重要环节，本节将讲解一些简单的调试技巧

以帮助读者朋友们更高效、更快速地开发程序。与 Java 开发不一样的是，在 Android 中控制台的信息相对很少，一些重要的信息我们都需要在 Log 日志中查找。

3.3.1　增加断点

与 Java 调试一样，选中你希望程序运行暂停的代码，双击代码的左侧就完成了断点的增加，让我们实际使用一下。例如，我希望程序在 setContentView()时暂停，那就在第 11 行代码左侧双击，之后会出现一个小圆点，这个小圆点就是所谓的断点了。同时，我们打开 Logcat，以便于观察程序的状态，如图 3.16 所示。

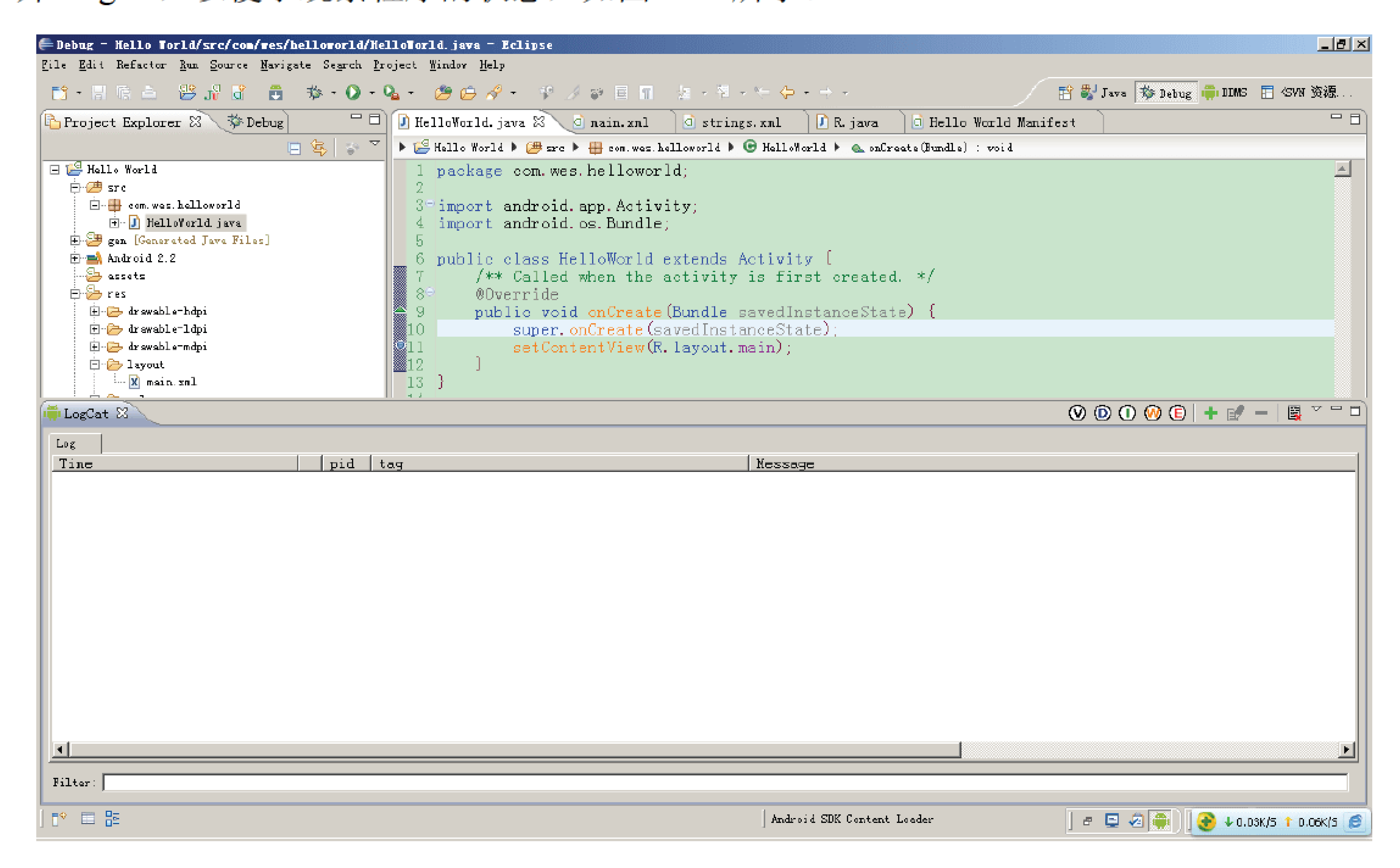


图 3.16　增加断点

这个时候 Logcat 还是一片空白，因为程序还没有进入调试阶段。

3.3.2　开始调试

在工程浏览器中选中你希望调试的工程，在右键菜单中选择 Debug as，再选择 Debug Configurations，如图 3.17 所示。

单击该选项，这个时候我们就进入了调试设置界面了，如图 3.18 所示。

在左侧选中 Android Application，其下选中要调试的应用，当然这些默认值都是设置好的，你并不需要去改动，接着单击 Debug 按钮就可以开始调试了。这个时候 Eclipse 会帮助我们启动模拟器，模拟器的启动可能会比较慢，这时因为系统需要对模拟器进行一些连接以便跟踪。

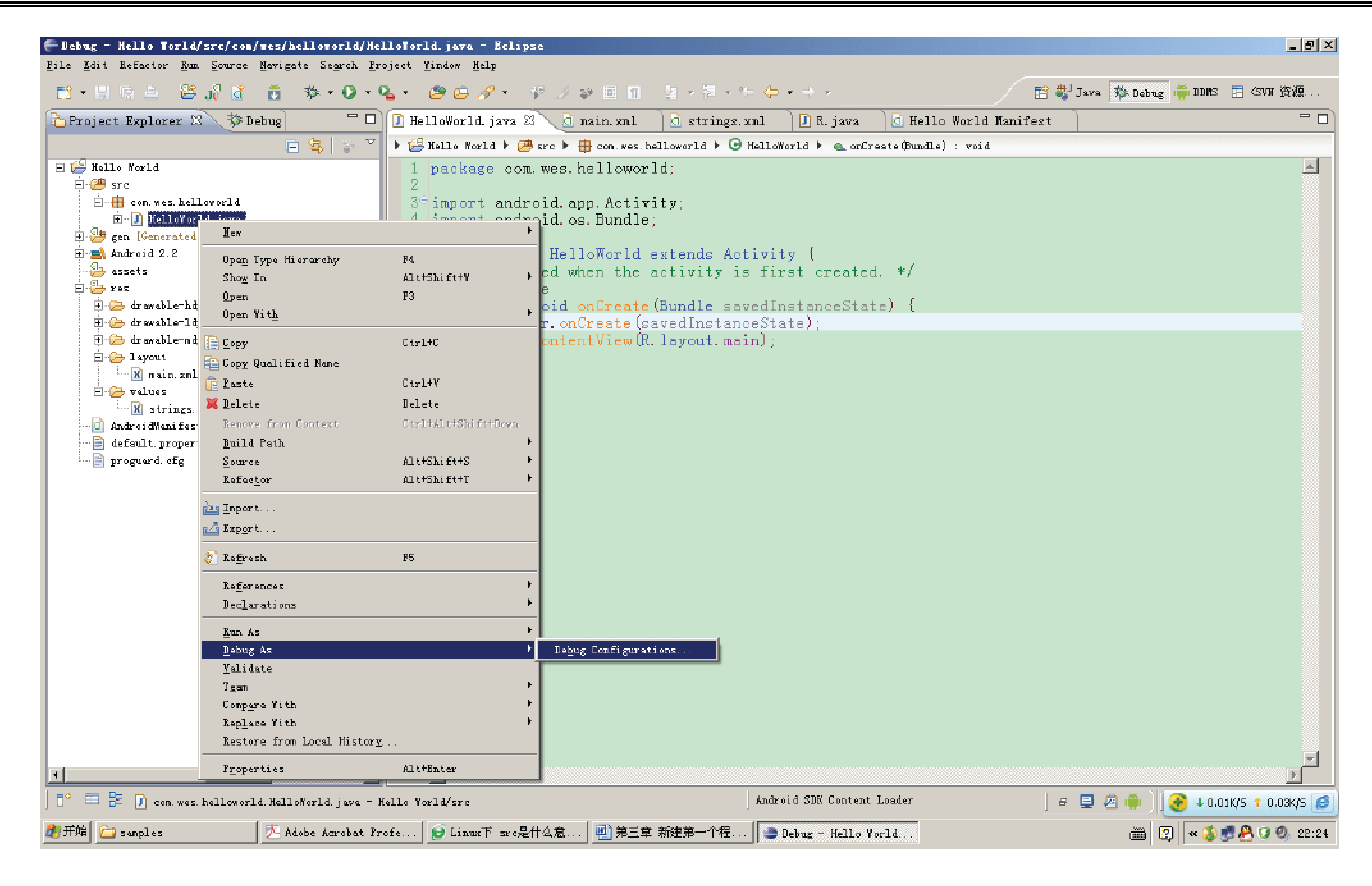


图 3.17　调试程序

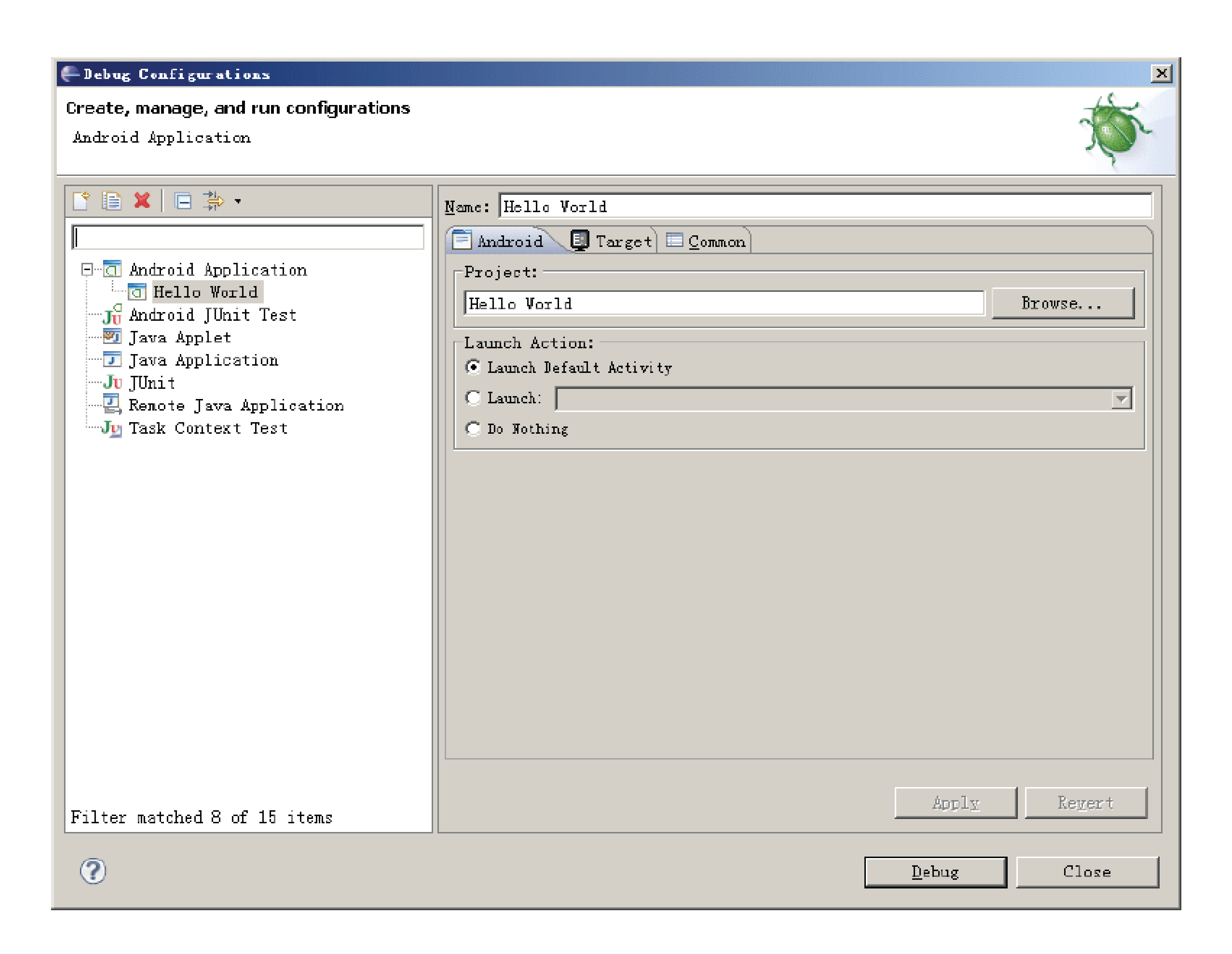


图 3.18　调试设置

3.3.3　单步调试

当程序运行到你设置断点的那一行代码时，程序会暂停，并在左侧显示一个小箭头以提示开发者程序运行的位置，左侧 Debug 框中显示了一些线程相关信息，而 Logcat 中则显示一些系统信息，如图 3.19 所示。

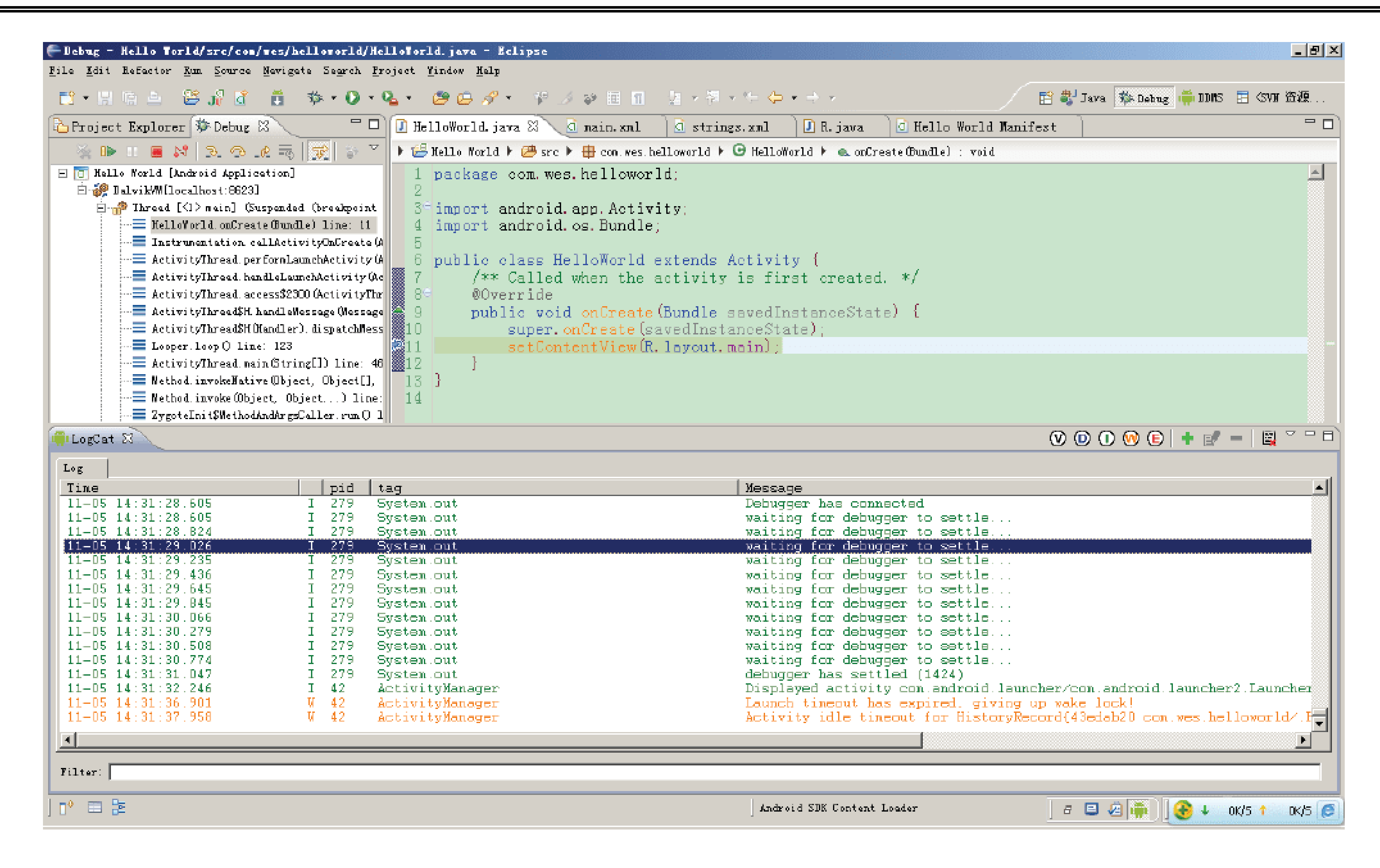


图 3.19　程序暂停

这个时候你有以下几种选择：

- F6：Step Over，按下 F6 键时程序向下走一行代码。
- F5：Step Into，按下 F5 键时进入该方法。
- F7：Step Return，按下 F7 键时返回上一行代码。
- F8：Resume，按下 F8 键时程序恢复运行，直到下一个断点。如果接下来没有断点则直接运行到程序结束，或运行到出现异常为止。

如果程序出现异常，在 Logcat 中可以看到相关信息。关于 Logcat 的使用在下一章中会有讲解。

3.4　更多示例程序

Android SDK 提供了一些范例程序以便开发者学习使用，它们在\<SDK 所在路径>\samples\android-8 中。通过阅读其中的一些程序可以学习更多 Android 开发所需的知识，本节将列举其中一些经典的范例程序，以供参考。

3.4.1　导入 Samples

Android SDK 中为我们提供的丰富的 Samples 可不能浪费了，那么怎么使用他们呢？

首先，按照新建工程的步骤进入如图 3.20 所示的新建 Android 工程对话框，然后选中 Create project from existing source，接着单击 Browse 按钮，进入图 3.21 所示的文件浏览窗口。

选择相应的资源，如选择 LunarLander。单击“确定”按钮，并选择一个 Android SDK 版本，单击 Finish 按钮完成创建。这时在工程浏览器中就可以看到 LunarLander 工程了。

展开工程，所有的文件夹都出现了，接下来还犹豫什么？赶紧运行一下看看效果吧，然后再分析分析代码，通过这些 Samples 迅速提升自己的 Android 编程水平！

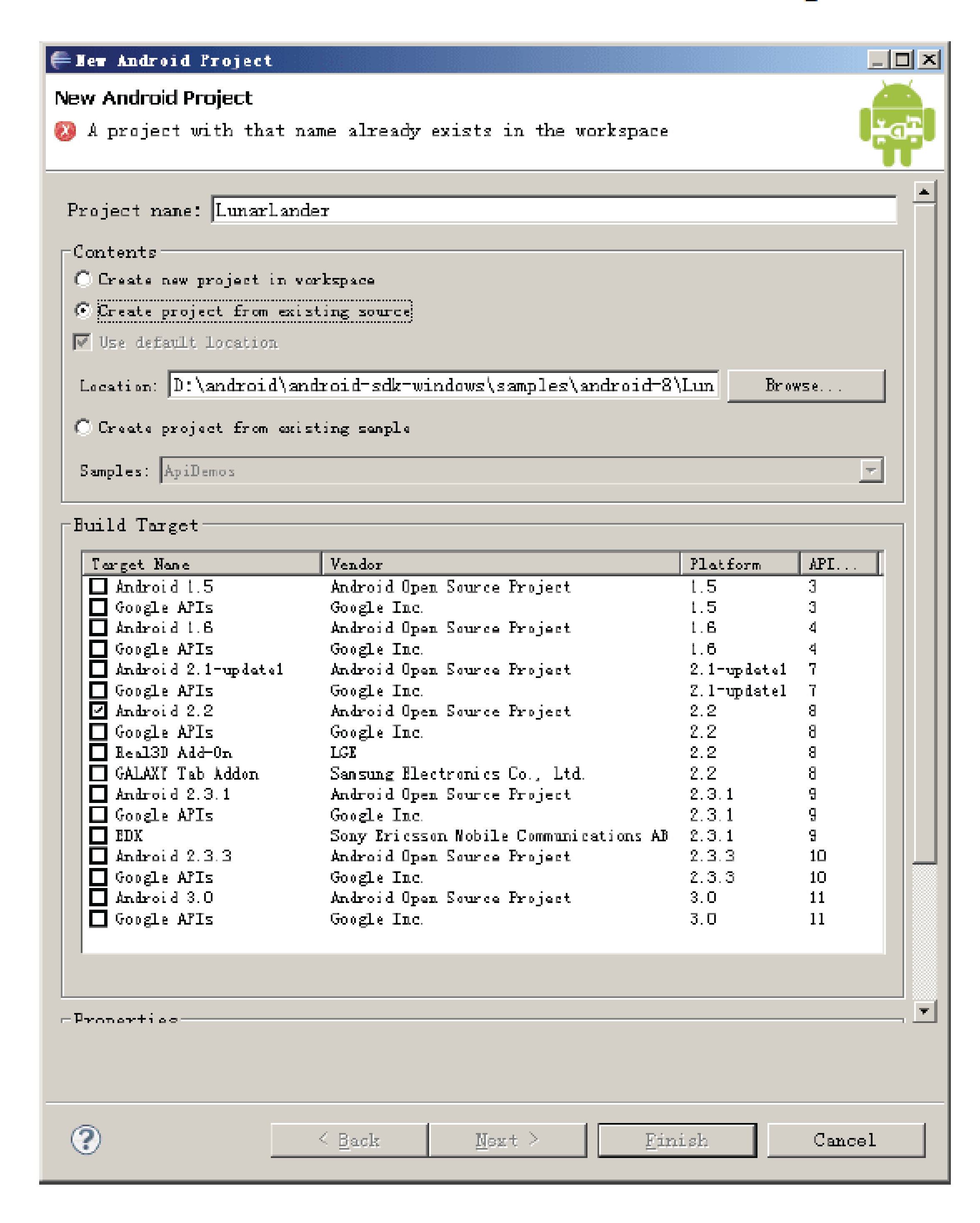


图 3.20　新建 Sample 工程

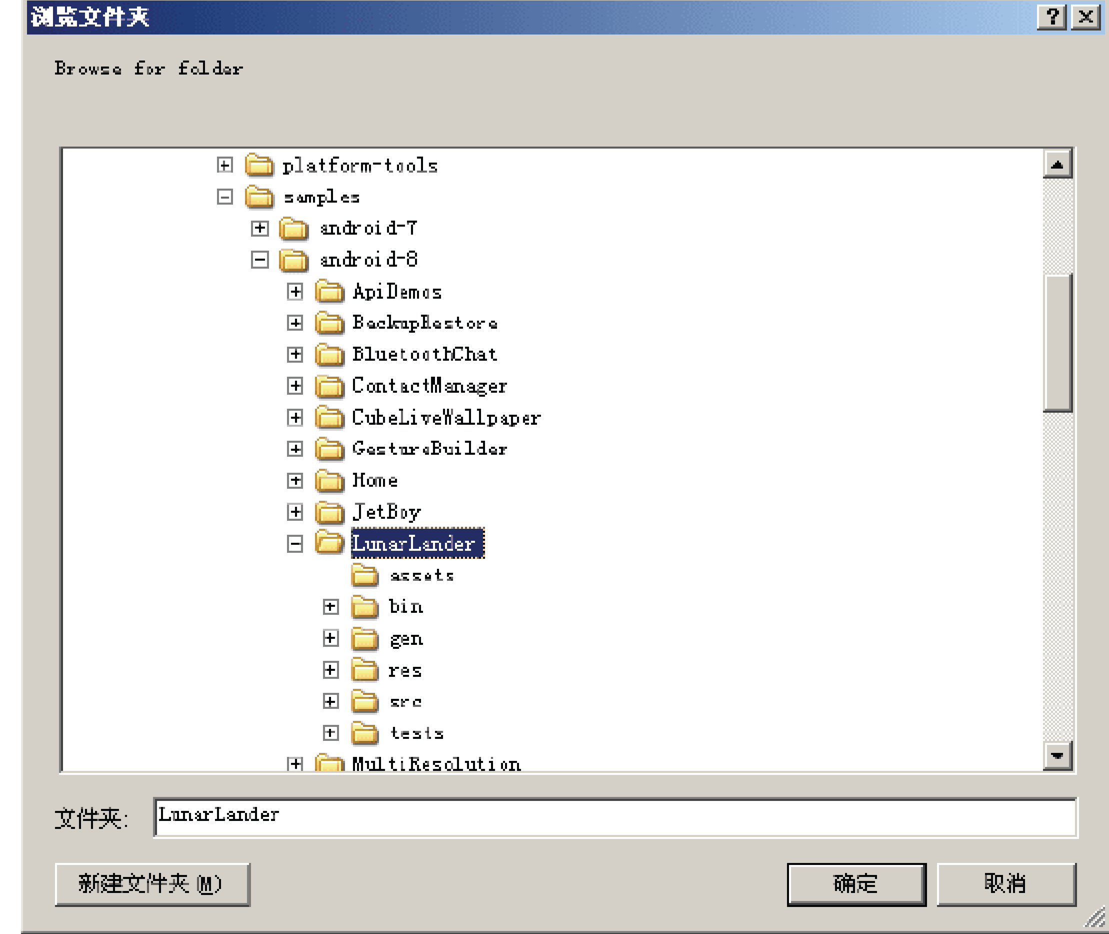


图 3.21　选择 samples 下的任意工程

3.4.2　经典范例

开发者最好能完全参透所有的 Android Samples，如果你觉得工作量太大，那么笔者就先推荐一些 Sample。

1．SkeletonActivity

该实例展示了一些基本组件的使用，如 TextView、EditText、ImageButton 等。组件是 Android 应用开发中重要的组成部分，界面开发全靠它，在第 5 章会有详细的讲解。运行效果如图 3.22 所示。

2．LunarLander

一个登月的小游戏，通过操作方向键和点火来使飞船能安全着陆。通过本例可以学习键盘快捷键的使用、菜单的使用、线程的使用等，运行界面如图 3.23 所示。

3．NotePad

记事本实例，界面本身没有什么需要注意的，需要学习的是该实例中关于数据库的使用，以及 ContentProvider 的使用。这部分内容在第 8 章 Android 数据存储中会有讲解。

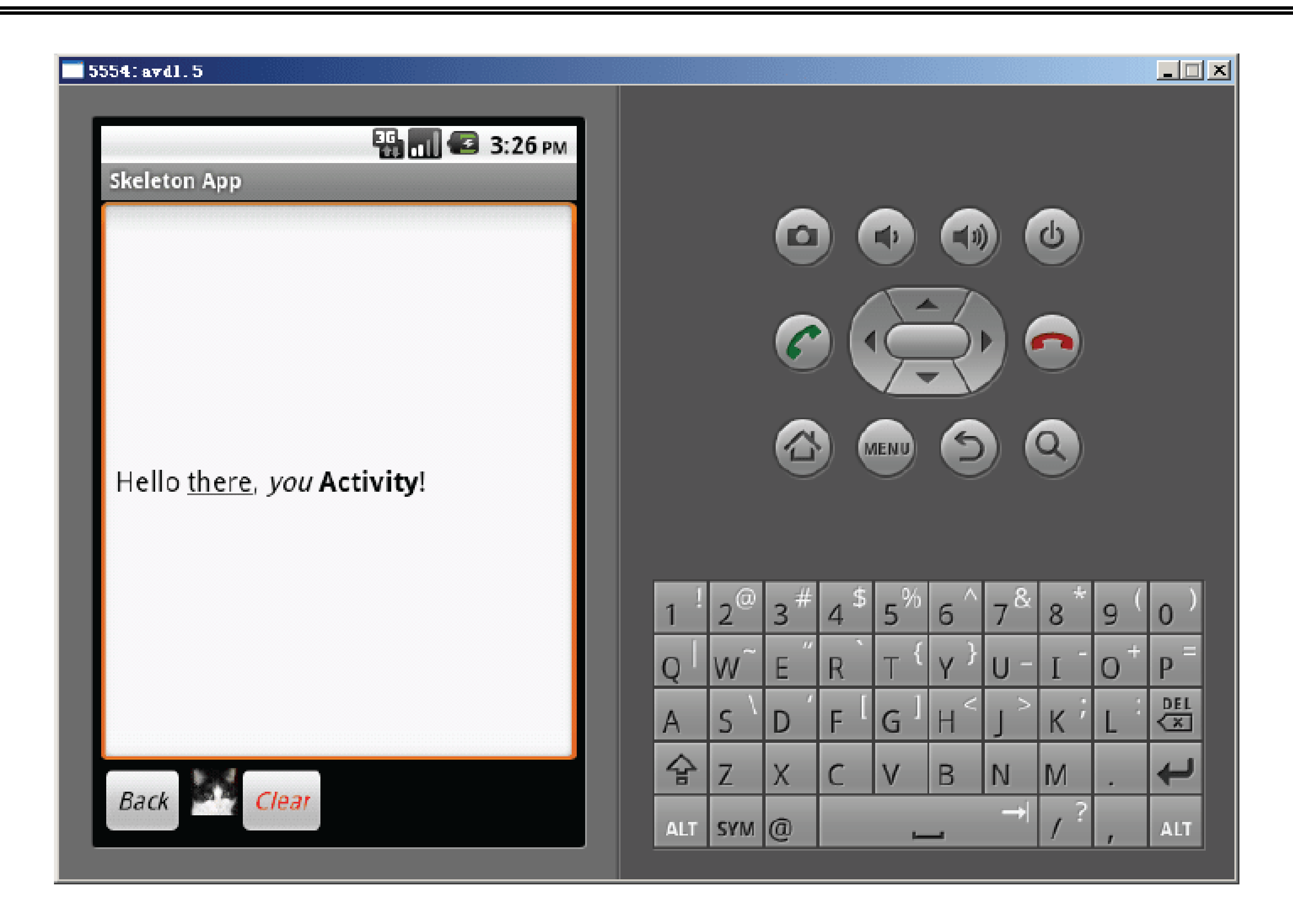


图 3.22　SkeletonActivity 运行中



图 3.23　游戏运行中

4．APIDemos

该实例中包含有很多 API 的演示，包括 app、ContentProvider、Graphics 等等。通过该 Sample 可以了解这些 API 的调用方法，运行效果如图 3.24 所示。

5．SoftKeyboard

通过本实例可以学习调用软键盘，如何将软键盘与单击事件绑定、添加软键盘的按键支持等等。这在实际的开发中还是比较实用的，运行效果如图 3.25 所示。

6．Snake

贪吃蛇，这款经典的游戏相信大家都不会陌生，通过本实例可以学习自定义组件的使用，其中有很多游戏开发需要使用的技术，故常被作为游戏开发的起点，运行效果如图 3.26 所示。

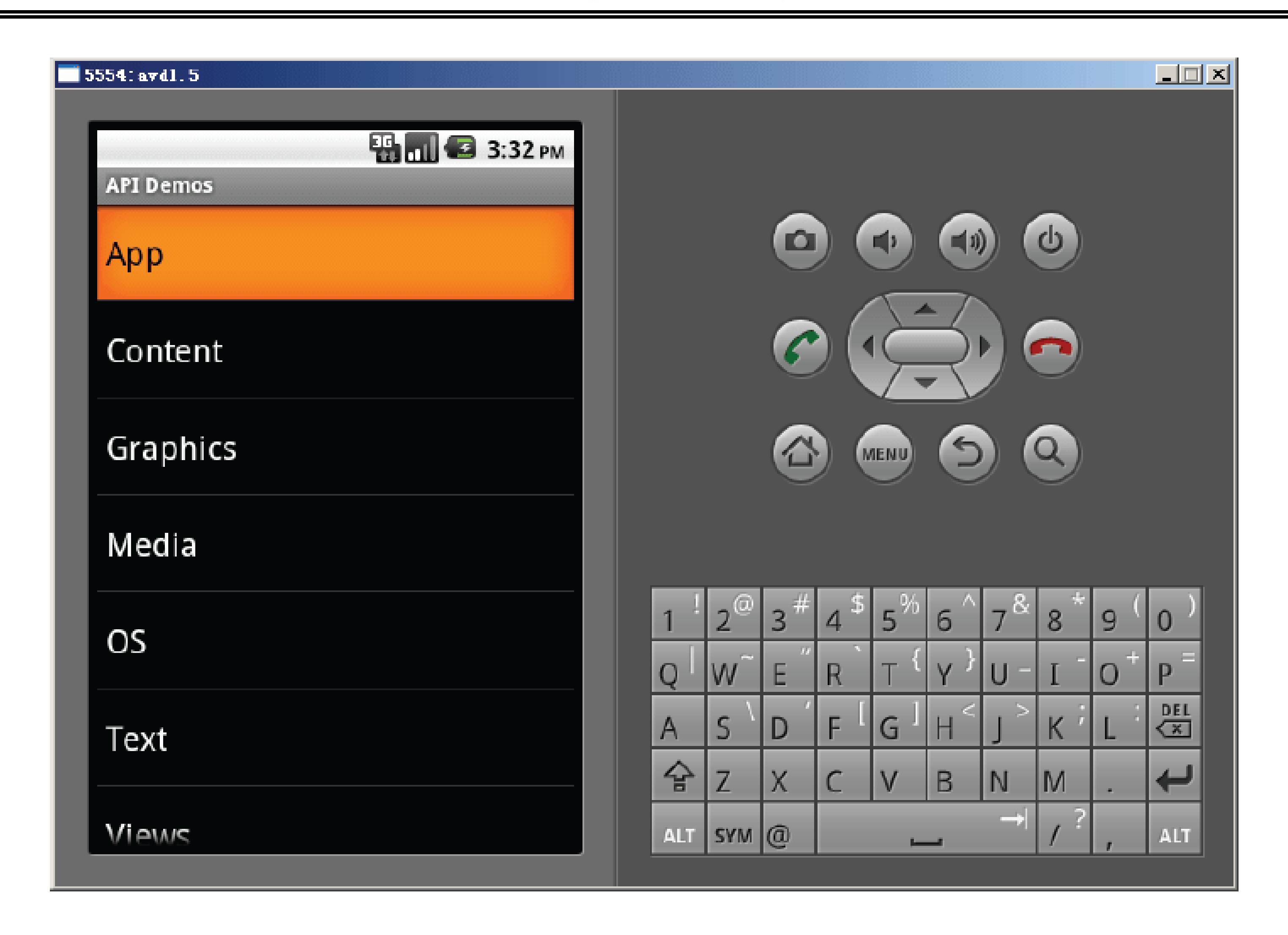


图 3.24　API Demos

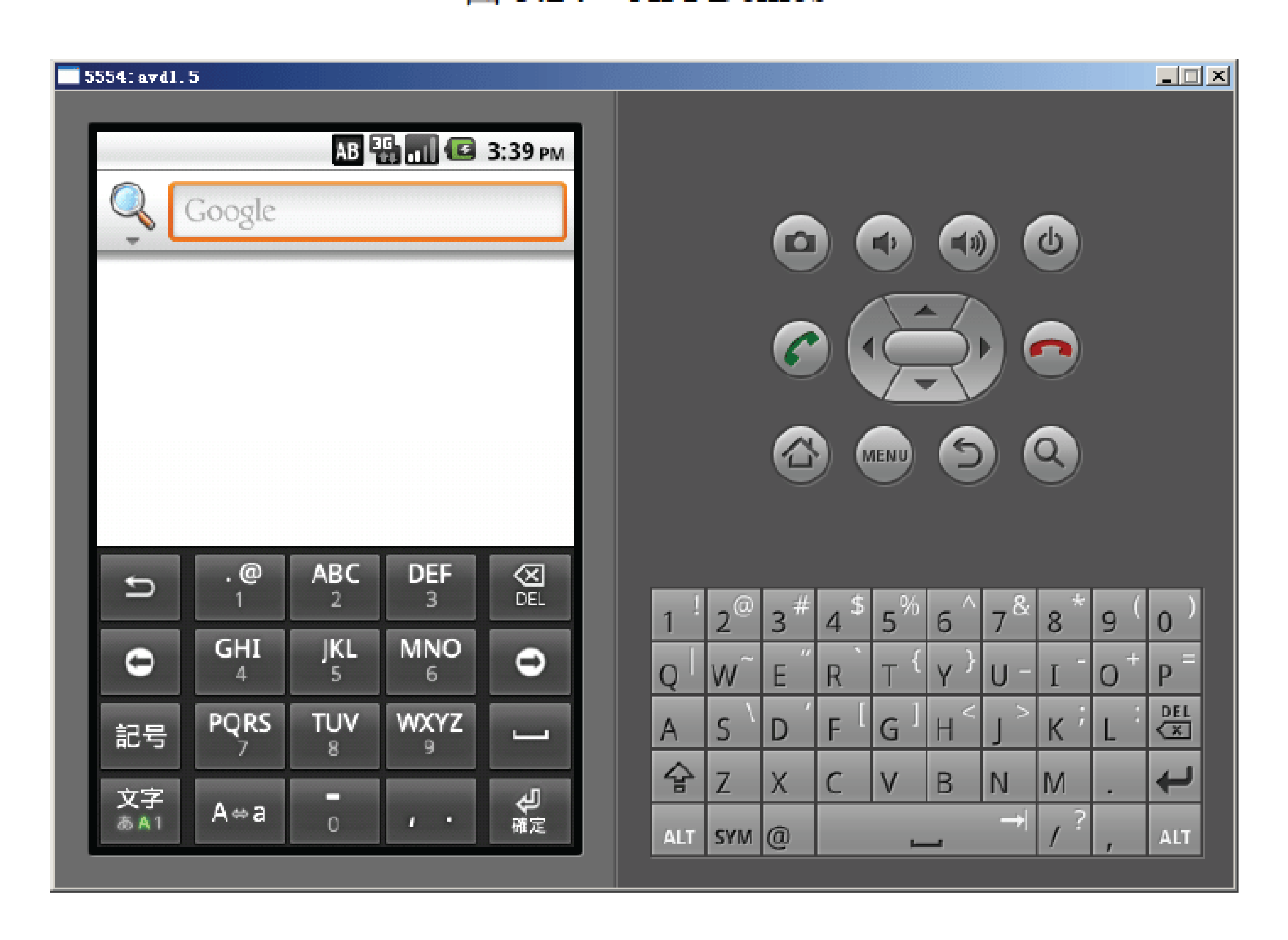


图 3.25　软键盘

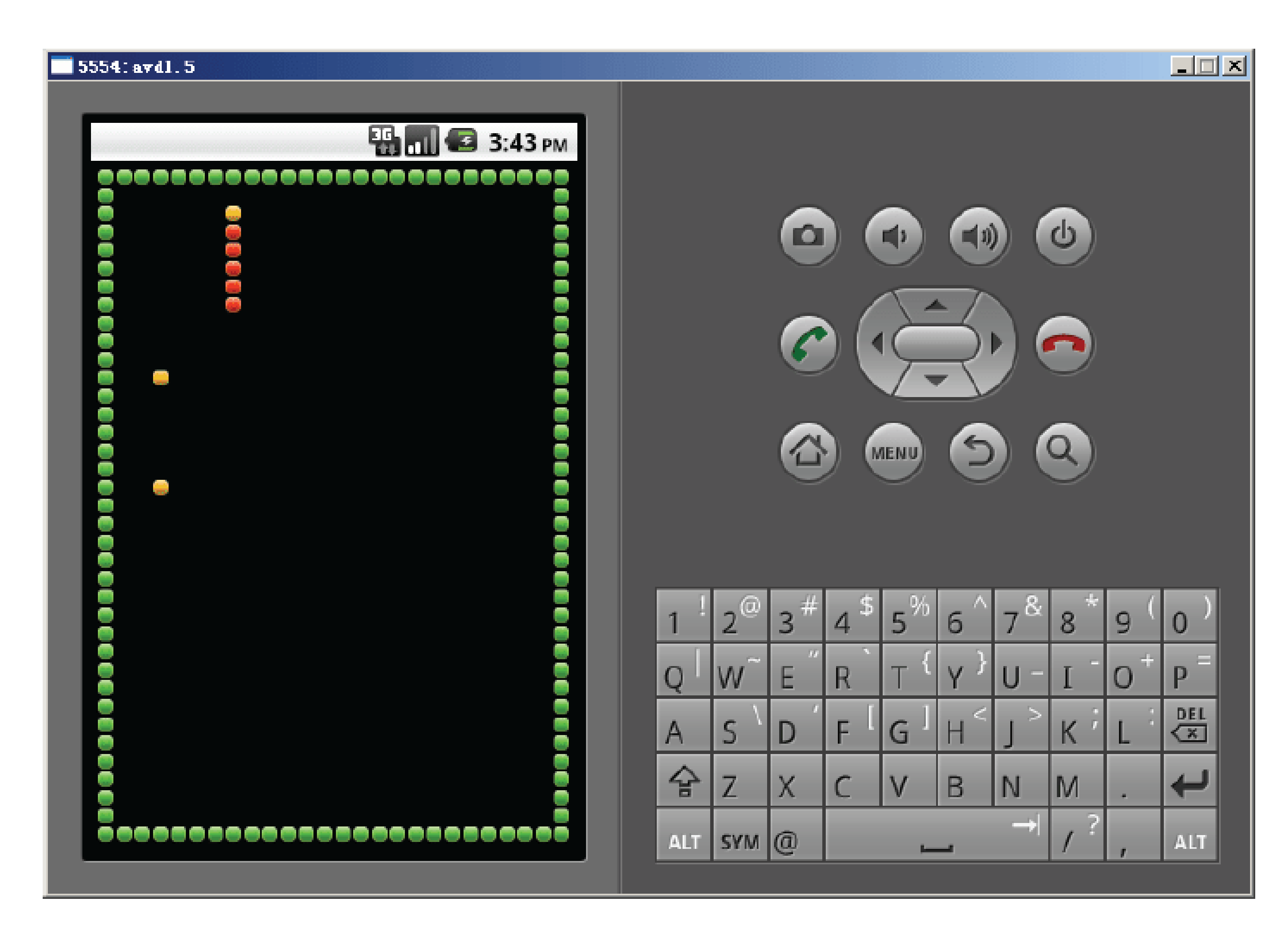


图 3.26　贪吃蛇进行中

7. Home

主题，该实例实现了主题的制作、注册和应用，通过该实例可以学习如何进行主题类的开发，其运行效果如图 3.27 所示。

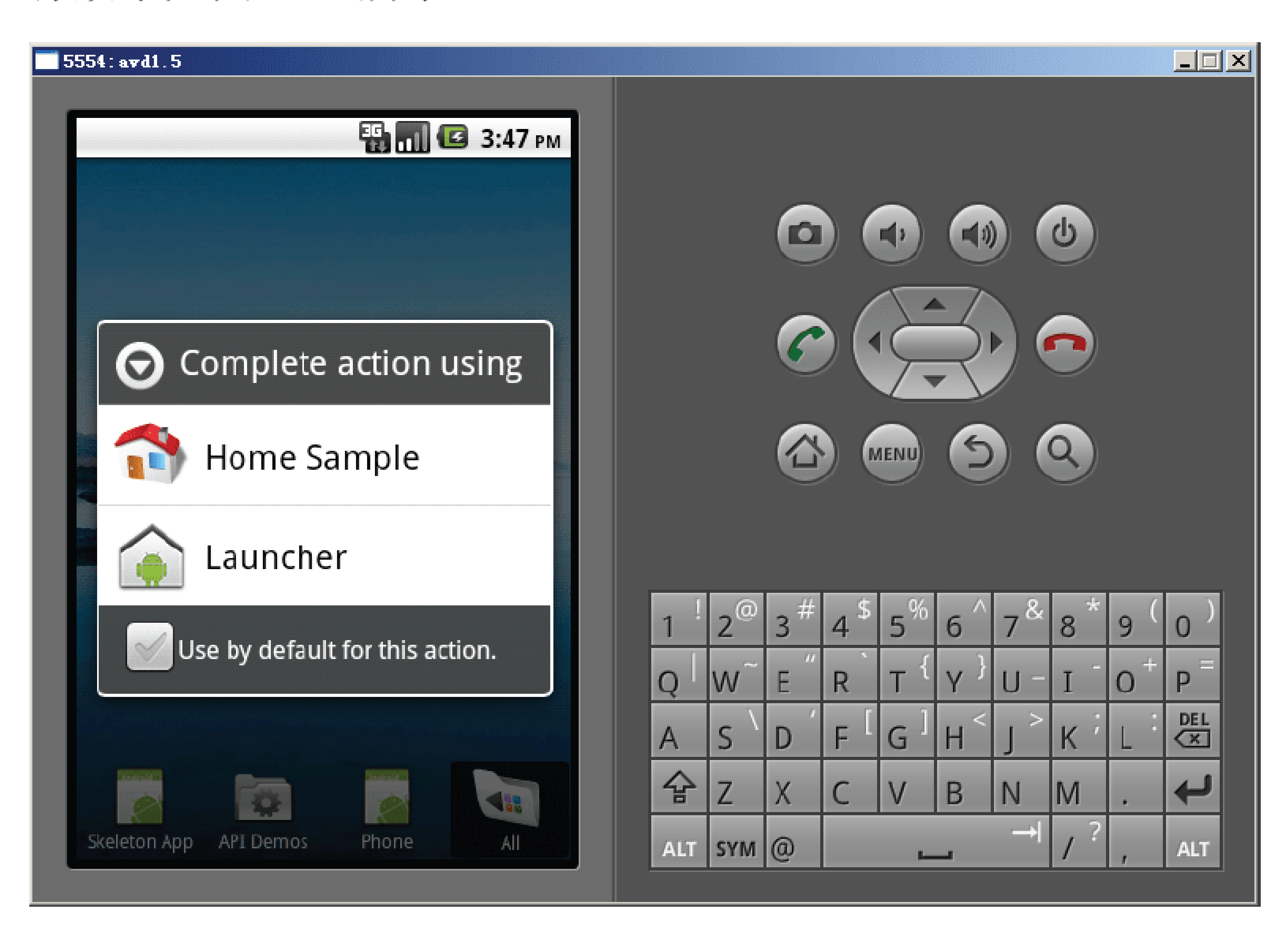


图 3.27　主题改变

就先推荐到这里了，最后还是要强烈推荐读者朋友们吃透这些 Samples，这对于我们的学习和开发是很有帮助的！

3.5　小　　结

本章通过 HelloWorld 工程讲解了在 Eclipse 中一个工程的创建、运行、阅读以及调试。其中重点是 Eclipse 的使用和工程目录的结构理解，难点是工程的调试和各个文件的作用。最后，还推荐了一些 Android SDK 自带的 Samples，通过这些范例读者可以更好地学习 Android。下一章我们将讲解 Android 开发时需要使用的一些工具以及程序的发布。

第 4 章　使用 Android 工具

在第 3 章我们已经学会了新建工程以及调试工程。在调试工程时有许多工具可以帮助我们，如 Android 调试桥（Android Debug Bridge，ADB）、Dalvik 虚拟机调试监控服务（Dalvik Debug Monitor Service，DDMS）以及数据库查看工具 sqlite3。本章将介绍这些工具的使用方法。

4.1　使用 DDMS

Dalvik 虚拟机调试监控服务（Dalvik Debug Monitor Service，DDMS）是一组实用工具的有机结合，开发者可以通过 DDMS 监视模拟器甚至是真实设备。它包括的工具有：任务管理器（TaskManager）、文件浏览器（File Explorer）、模拟器控制台（Emulator console）以及日志控制台（Logging console）。

4.1.1　认识 DDMS

首先打开第 3 章中新建的 HelloWorld 程序，并运行它。接着再运行 DDMS 来观察程序的运行状况。打开 DDMS 的方法为：在 Eclipse 中选择 Windows 菜单中的 OpenPerspective，在弹出的子菜单中选择 DDMS 就可以了，如图 4.1 所示。

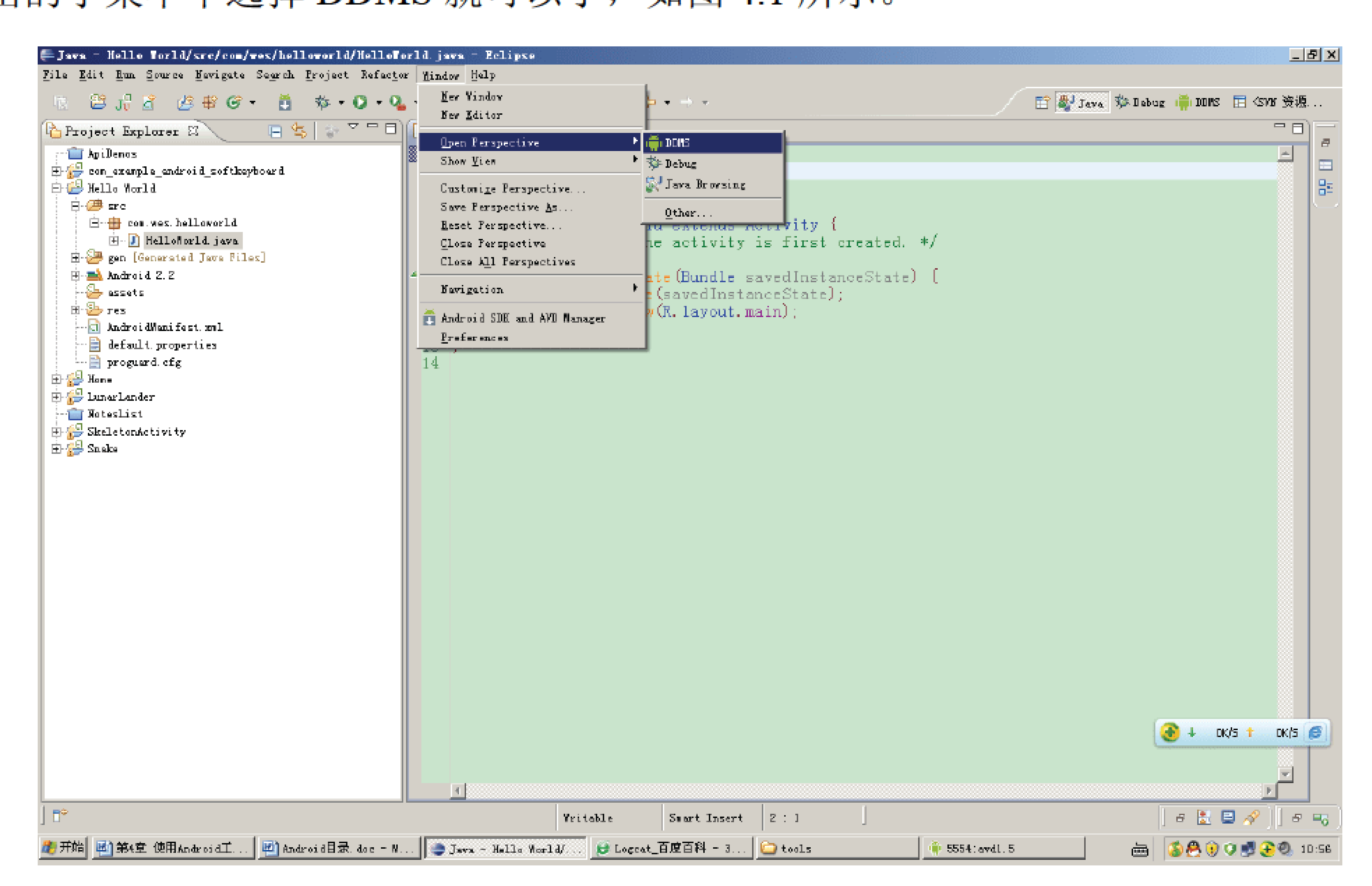


图 4.1　打开 DDMS

单击 DDMS 后，我们可以看到如图 4.2 所示的 DDMS 使用界面。

图 4.2　DDMS 使用界面

在 DDMS 中，你可以：

（1）查看设备列表，以及各个设备的运行状态。

（2）通过 Logcat 查看程序的日志记录。

（3）通过文件浏览器 File Explorer 查看并操作设备上的文件。

（4）查看每个进程或线程状态；触发 Java 的垃圾回收（GC）；查看应用程序使用的堆；同样可以终止线程。

（5）捕捉屏幕，通过 Screen Capture 可以很方便地捕捉模拟机或者真实设备的屏幕画面。

（6）模拟发送 GPS、模拟来电等。

4.1.2　使用进程

我们知道每个 Android 应用程序都运行在操作系统的单独虚拟机（VM）中，并且每个程序都用其包名作为 Id。

通过 DDMS 左侧的面板我们可以查看所有正在设备上运行的 VM 实例，它们的名字都是自己的包名。例如，我们可以找到正在运行的 Id 为 com.wes.helloworld 的 VM 实例。当然目前我们只能看到它正在运行却不知道其具体的状态如何，如果 DDMS 的功能仅仅是这样那肯定是不够强大的，接下来我们就继续深入地使用它。

1. 关联调试器

关联调试器的具体步骤为：

（1）在左侧的设备面板中选中你要调试的包名，使其高亮。

（2）单击上方的绿色小虫标志开始调试。

单击后，我们就完成了调试器的关联。接下来我们可以查看线程。

2. 查看线程

依然选中要调试的包名使其高亮，接着单击上方的 3 个向右的箭头图标，该按钮名为 update threads。这时在右侧面板的 Threads 标签中就可以看到该进程中运行的一系列线程了，如图 4.3 所示。

图 4.3　查看所有线程

如果仅仅如此，那么我要说 DDMS 仍然不够强大，如果能够再进一步进入到线程的内部查看正在运行的方法就更好了。当然，DDMS 肯定能够做到。你只需：

（1）打开 Threads 标签页。

（2）选中你要查看的线程。

（3）单击 Refresh 按钮。

这个时候就可以在 Threads 标签页的下方面板中看到该线程中运行的方法以及各个类了，如图 4.4 所示。

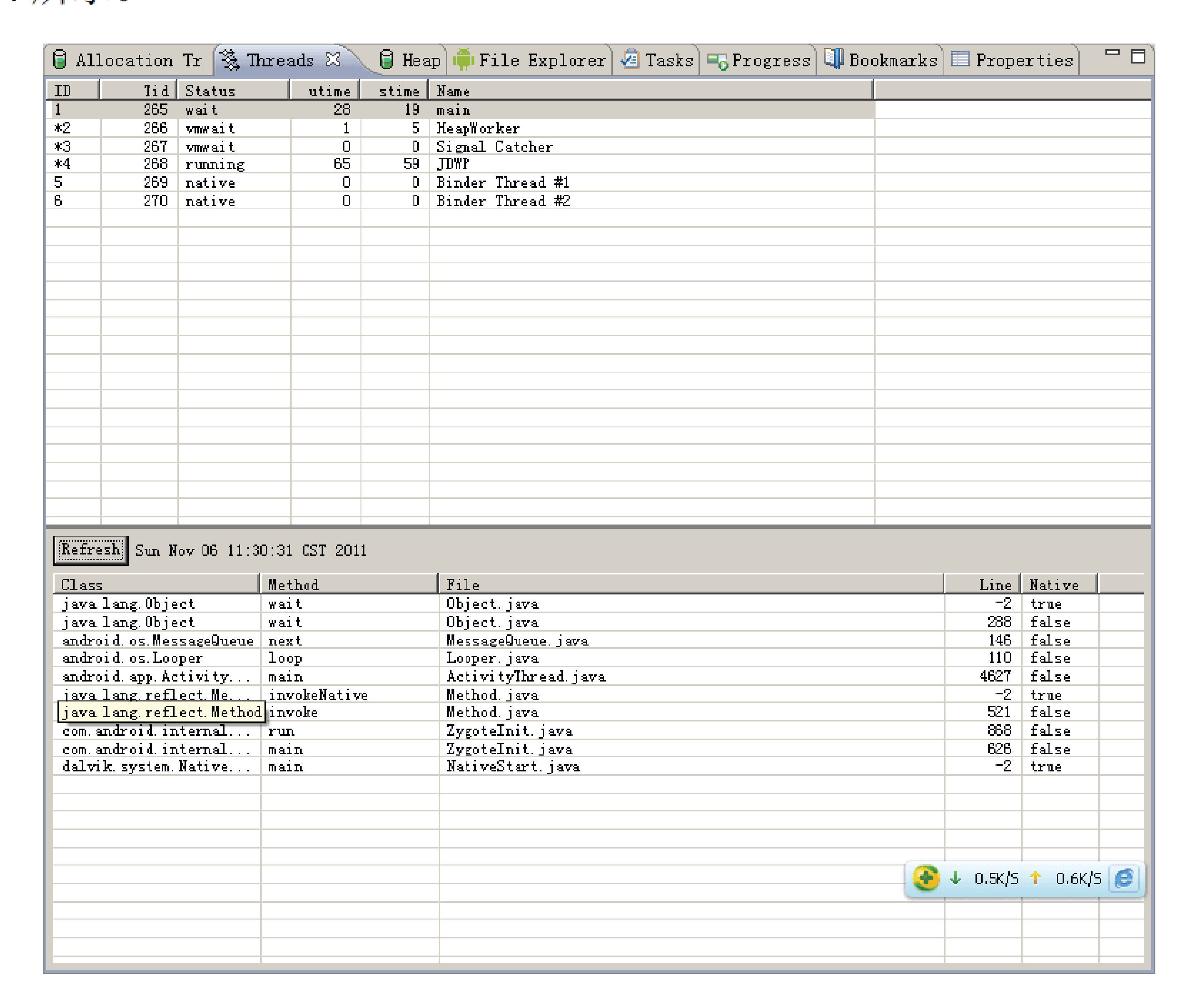


图 4.4　Refresh 线程

3. 查看堆统计

使用 DDMS 甚至可以查看应用程序中堆的统计数据。查看时需要执行的步骤为：

（1）在左侧面板中找到要查看的包，选中它。

（2）单击绿色的小桶图标按钮，该按钮的名字是 Update Heap。这时数据将显示在右侧的 Heap 标签页中。也许这个时候还没有任何数据显示，不要着急，单击一下 Cause GC 按钮就可以看到数据出现了。这是因为 Heap 标签页是在每次 GC 之后才会刷新数据，除了被动等待垃圾回收（GC）以外，我们可以通过单击刚才的 Cause GC 按钮主动触发垃圾回收。

这个时候，在右侧的 Heap 标签页中显示如图 4.5 所示。

Allocation Tr | Threads | Heap | File Explorer | Tasks | Progress | Bookmarks | Properties

Heap updates will happen after every GC for this client

ID	Heap Size	Allocated	Free	% Used	# Objects
1	2.754 MB	2.257 MB	508.74...	81.96%	46,693

Cause GC

Display: Stats

Type	Count	Total Size	Smallest	Largest	Median	Average
free	399	482.211 KB	16 B	239.578 KB	64 B	1.208 KB
data object	29,624	887.336 KB	16 B	552 B	32 B	30 B
class object	2,161	355.047 KB	168 B	176 B	168 B	168 B
1-byte array (byte[], boolean[])	1,368	224.234 KB	24 B	2.148 KB	32 B	167 B
2-byte array (short[], char[])	10,854	622.297 KB	24 B	15.523 KB	48 B	58 B
4-byte array (object[], int[], f...	2,559	214.422 KB	24 B	16.023 KB	40 B	85 B
8-byte array (long[], double[])	127	7.539 KB	32 B	2.000 KB	32 B	60 B
non-Java object	231	26.781 KB	16 B	8.023 KB	32 B	118 B

图 4.5　显示堆数据

（3）选中任意对象，它的使用状况将会以图表的形式显示在下方的面板中，如图 4.6 所示。

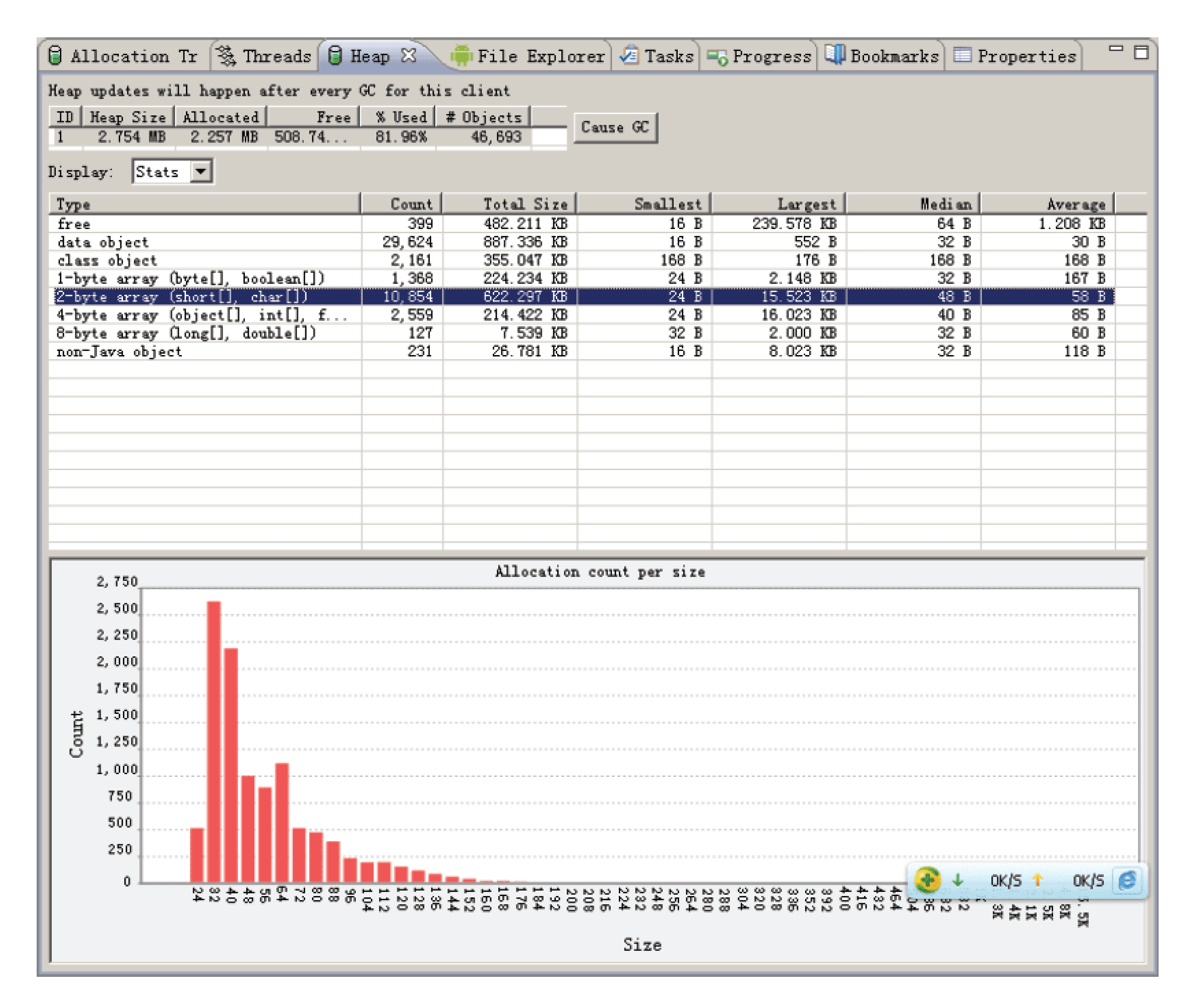


图 4.6　查看具体对象

4. 终止进程

终止进程的方法为：

（1）选中你要终止的进程。

（2）单击红色的停止图标按钮，该按钮的名字是 Stop Process。

单击后该进程则被终止，调试结束。

4.1.3　使用文件浏览器

文件浏览器可以帮助我们很方便地查看模拟器或者设备上的文件，我们可以使用它将文件从手机导入到电脑，或将文件从电脑发送到手机。打开文件浏览器的方法为：

（1）选中你要查看的设备，使其高亮。

（2）选择 Window 菜单中的 Show View，接着选择 File Explorer。

操作示意如图 4.7 所示。

图 4.7　打开文件浏览器

这时，我们就可以看到文件浏览器的标签页了，将文件夹展开，显示如图 4.8 所示。

1. 从手机上拷贝文件

如果希望从手机设备上将文件拷贝到电脑上，只需如下 3 个步骤：

（1）选中你希望操作的文件。

（2）单击文件浏览器标签页右上角的向左箭头图标，该图标名为：Pull a file from the decvice。

（3）在弹出的浏览窗中选择文件的保存地址，确定后单击“保存”按钮就可以了，如

图 4.9 所示。

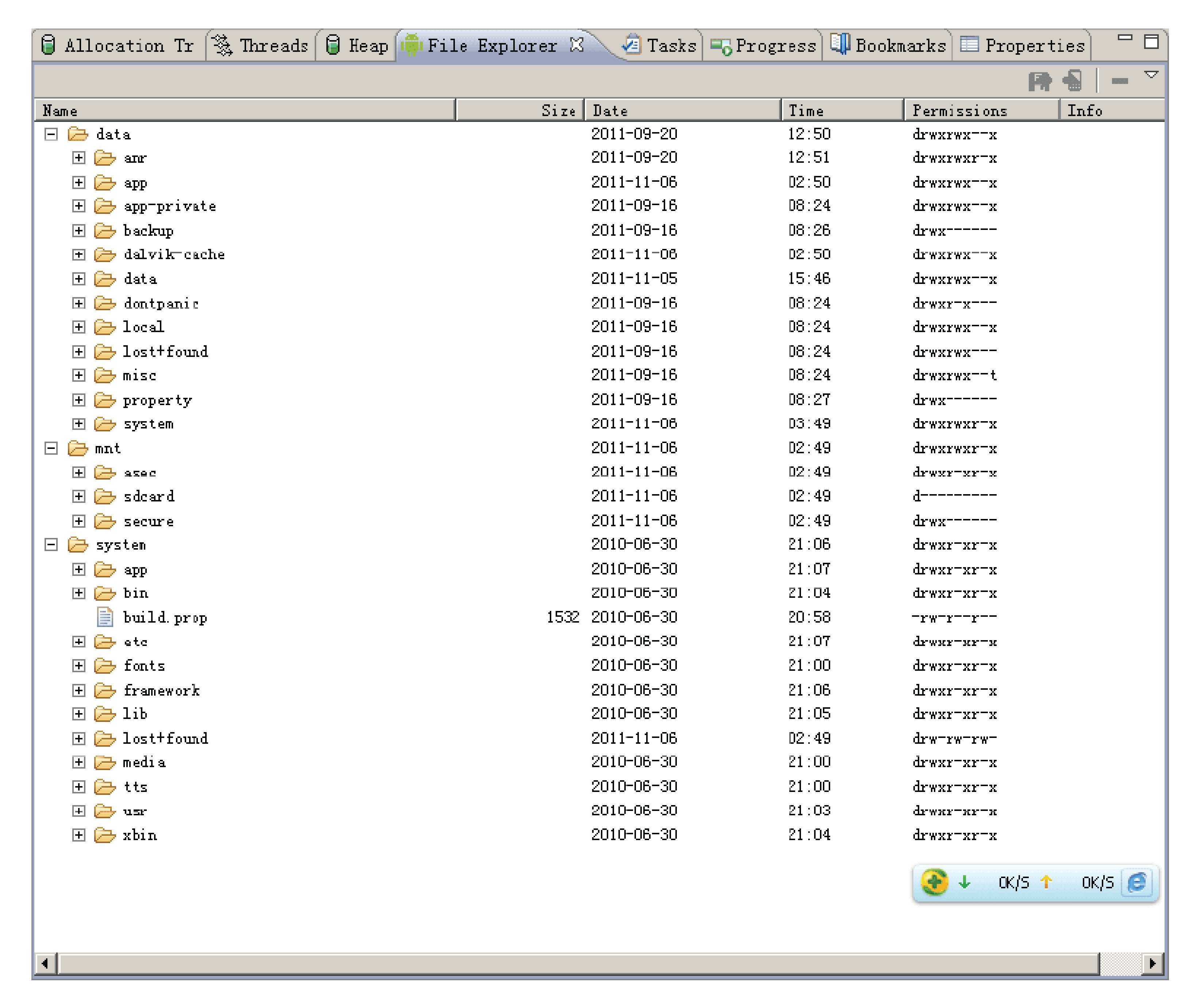


图 4.8　文件浏览器

2. 从电脑上拷贝文件到手机

拷贝文件到手机上时，同样需要如下 3 个步骤：

（1）在文件浏览器中选择你希望保存文件的文件夹，使其高亮。

（2）单击文件浏览器标签页右上角的向右箭头图标，该图标名为：Push a file onto the device。

（3）在弹出的浏览窗口中选择目标文件，选中后单击打开，如图 4.10 所示。

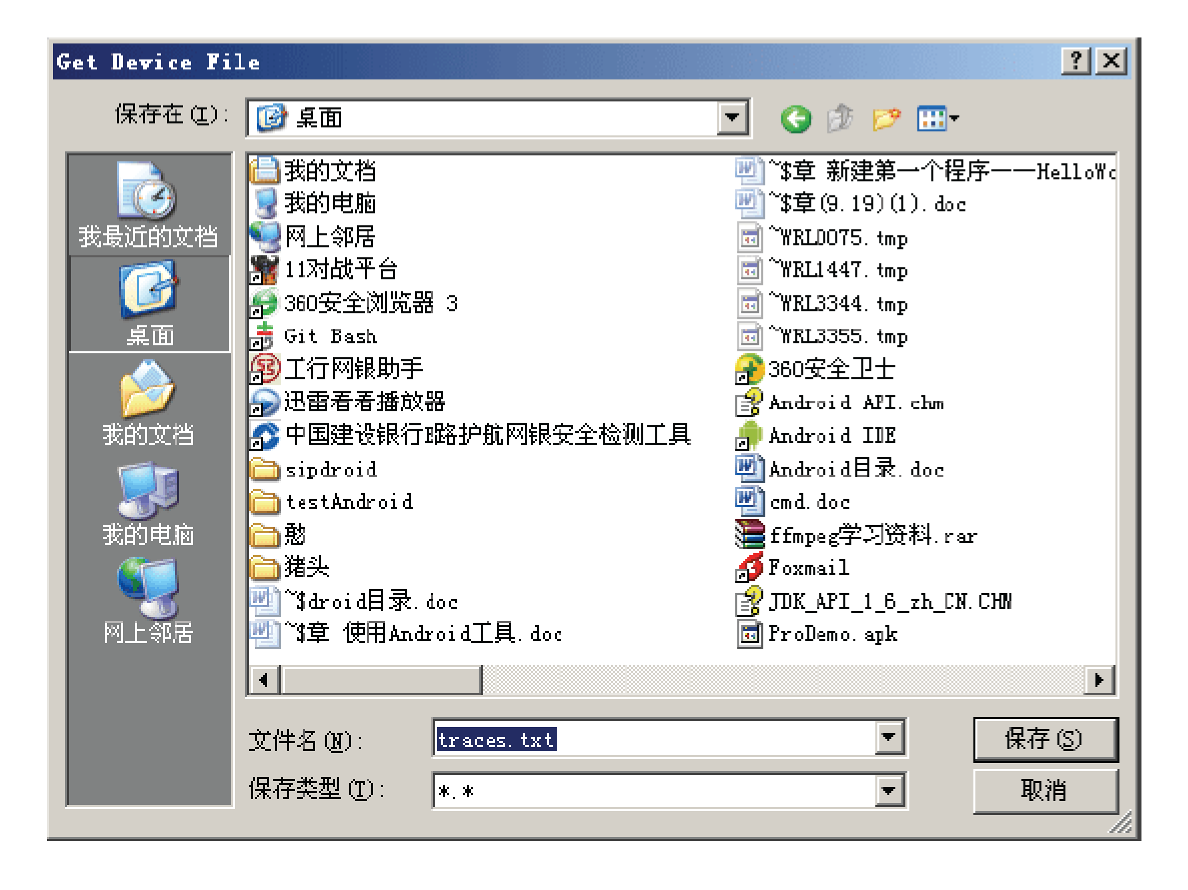


图 4.9　Get Device File 对话框

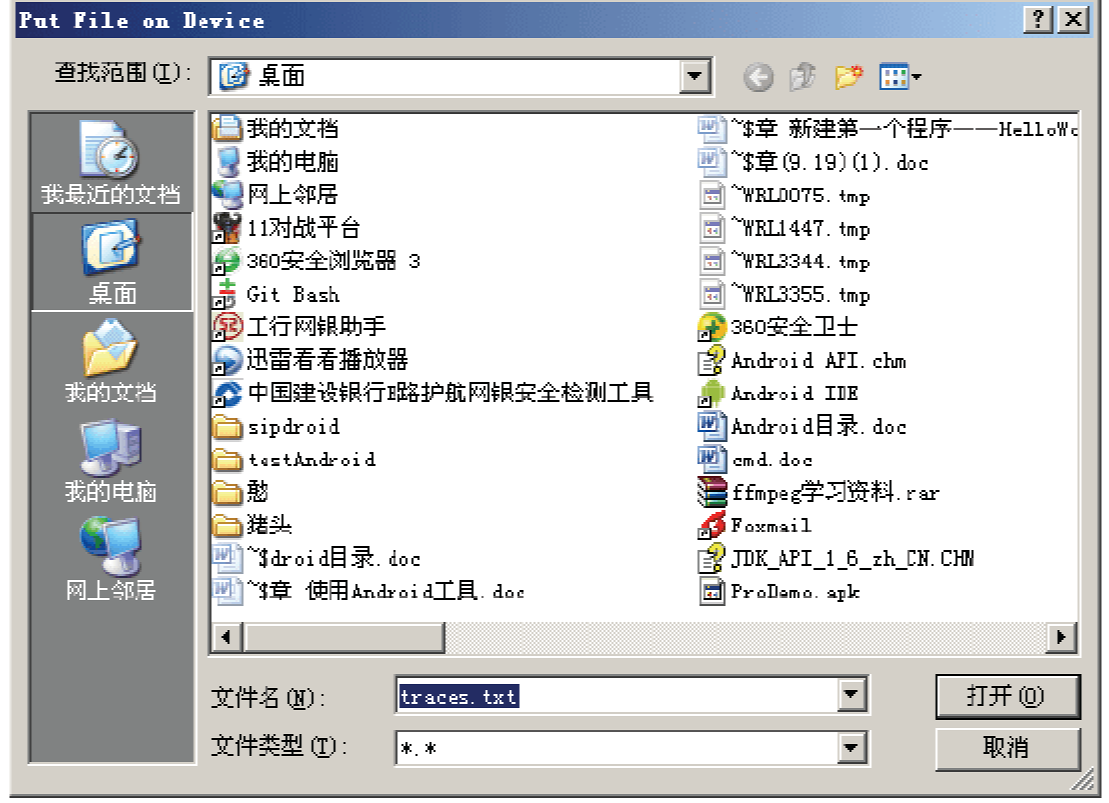


图 4.10　Put File on Device 对话框

3. 删除文件

目前文件浏览器只支持删除文件而不支持删除整个文件夹。删除文件时，步骤如下：

（1）选中你要删除的文件，使其高亮。

（2）在文件浏览器右上角单击红色的横线符号。

需要注意的是，该操作没有任何提示，所以执行时需要小心确认以防止误删，因为被删除的文件是没有办法恢复的。

4.1.4　使用模拟器控制

使用模拟器控制可以对模拟器进行操作，模拟以下状态：

（1）模拟语音来电。

（2）模拟接收短消息。

（3）模拟发送 GPS 信号。

接下来我们一一模拟这些常见情况：

1. 模拟语音来电

需要模拟语音来电时需要按照以下步骤：

（1）在 DDMS 的左侧面板中选中你需要操作的模拟器。

（2）在 Emulator Control 面板中的 Telephony Actions 菜单下的 Incoming number 编辑框中输入任意号码。

（3）选择 Voice 选项。

（4）单击 Call 按钮。

（5）使用 Hang up 可以挂起，图 4.11 显示了模拟时的操作界面。

当有模拟成功时，模拟器显示如图 4.12 所示的来电显示。

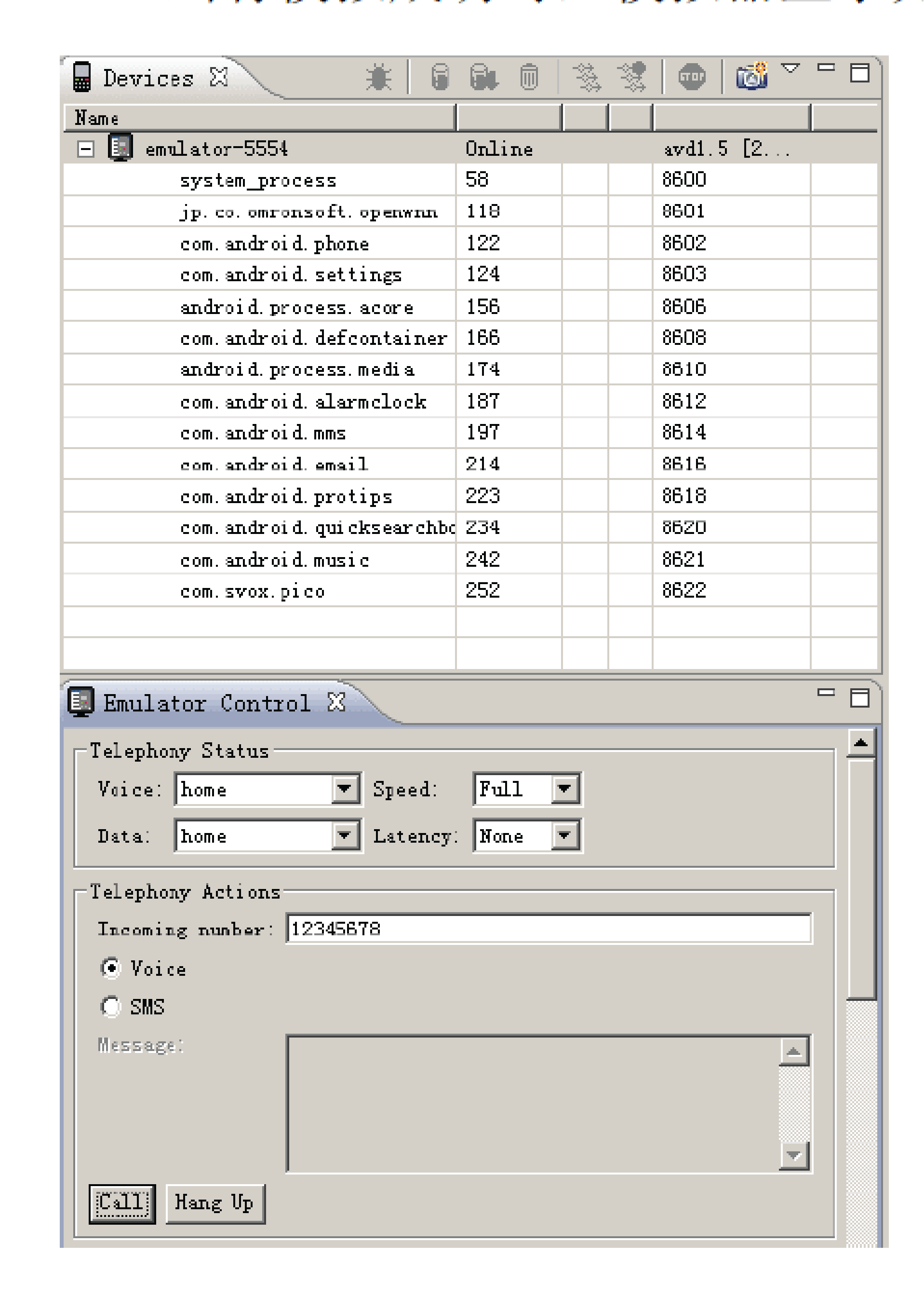


图 4.11　模拟器控制

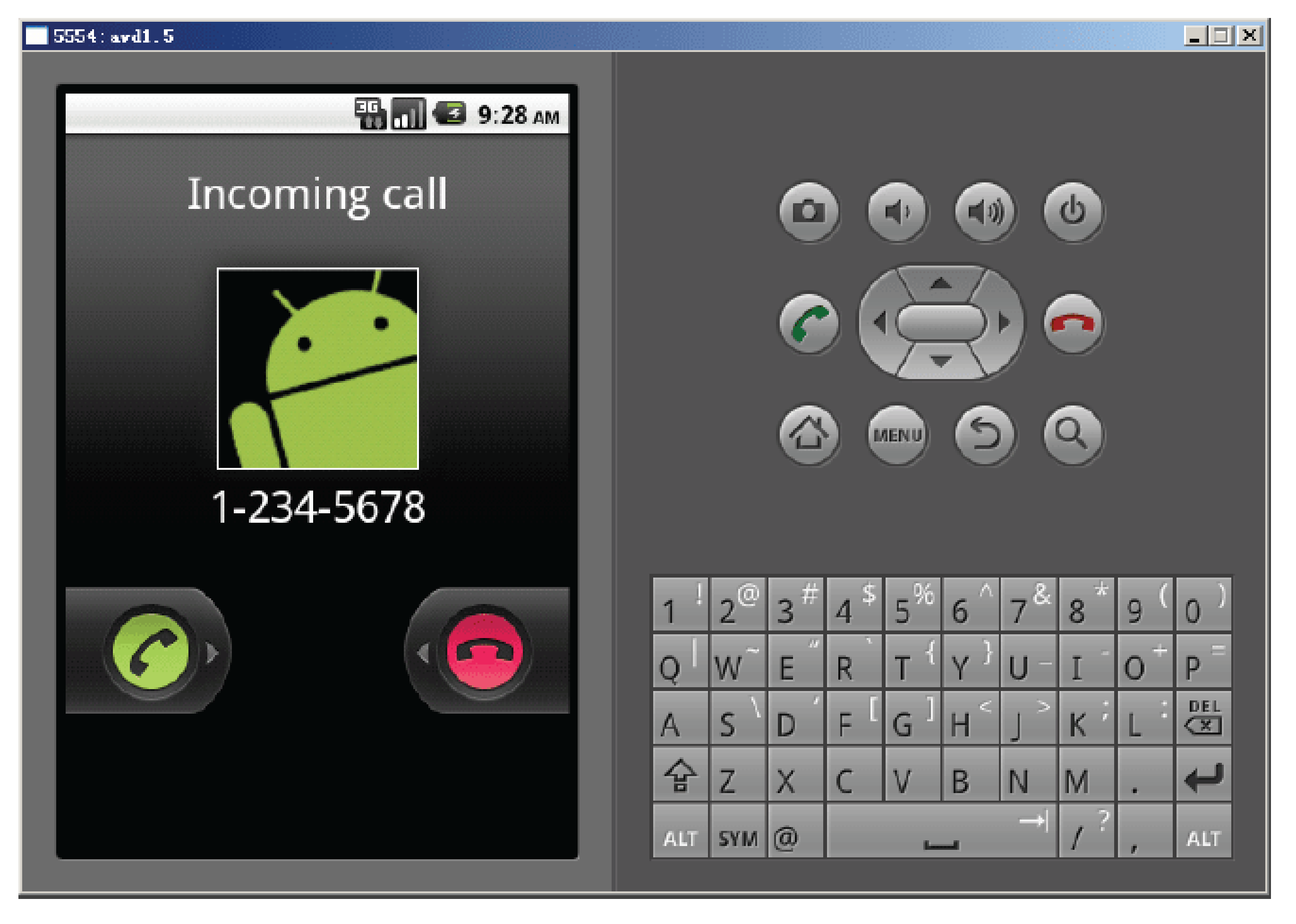


图 4.12　模拟来电

2. 模拟接收短信息

需要模拟接收短信息时需要按照以下步骤：

（1）在 DDMS 的左侧面板中选中你需要操作的模拟器。

（2）在 Emulator Contaol 面板中的 Telephony Actions 菜单下的 Incoming number 编辑框中输入任意号码。

（3）选择 SMS，在 Message 对话框中填入模拟的短消息内容。

（4）单击 Send 按钮模拟发送。

（5）模拟器接收到短消息时显示如图 4.13 所示。

3. 模拟发送GPS信息

模拟发送 GPS 信息需要以下几个步骤：

（1）在 DDMS 的左侧面板中选中你需要操作的模拟器。

（2）在 Emulator Contaol 面板中下拉按钮，直到 Location Controls。

（3）在 Longitude 与 Latitude 编辑框中分别输入经度和纬度。

（4）单击 Send 按钮发送 GPS 信号，操作如图 4.14 所示。

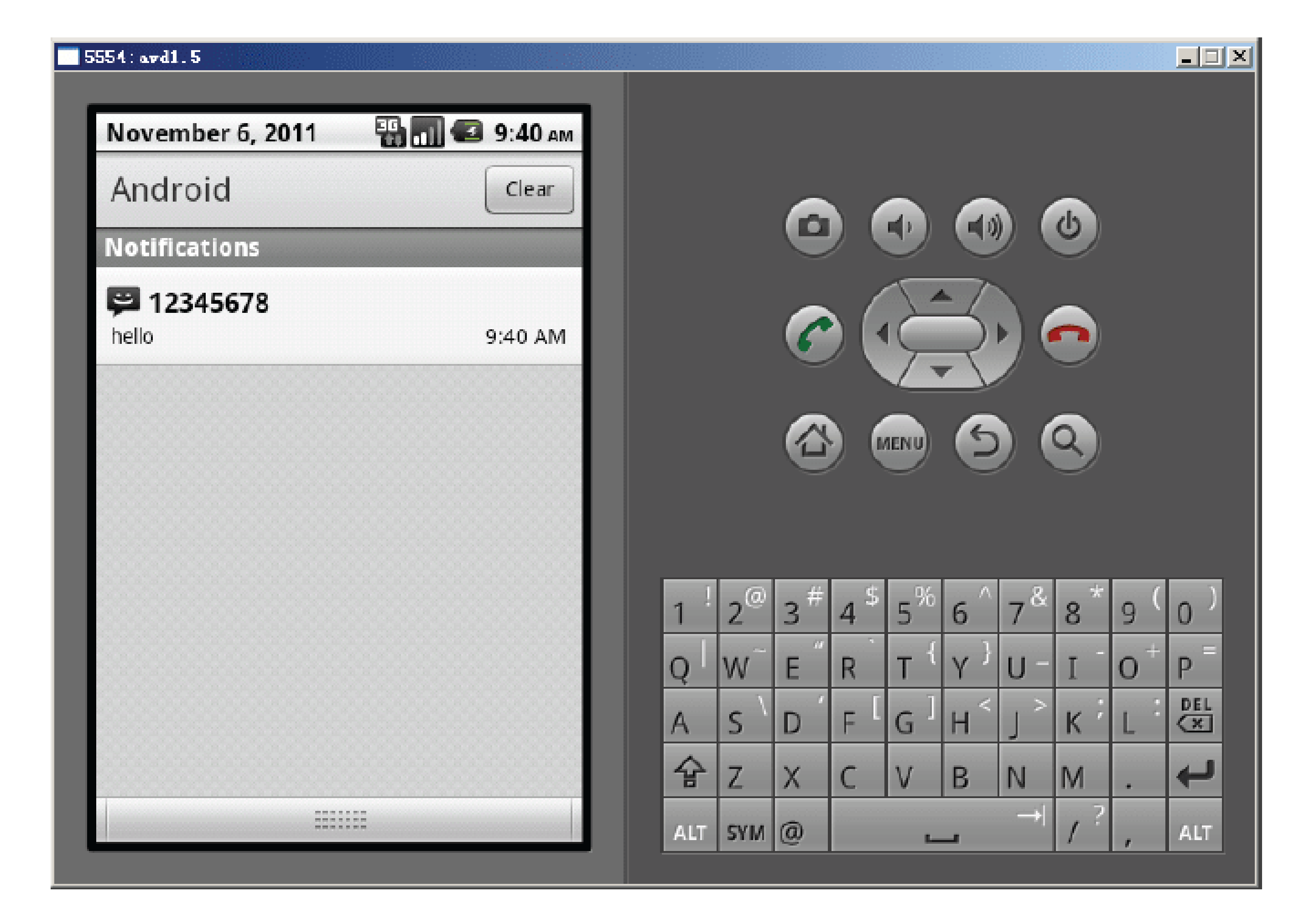


图 4.13　模拟接收短信

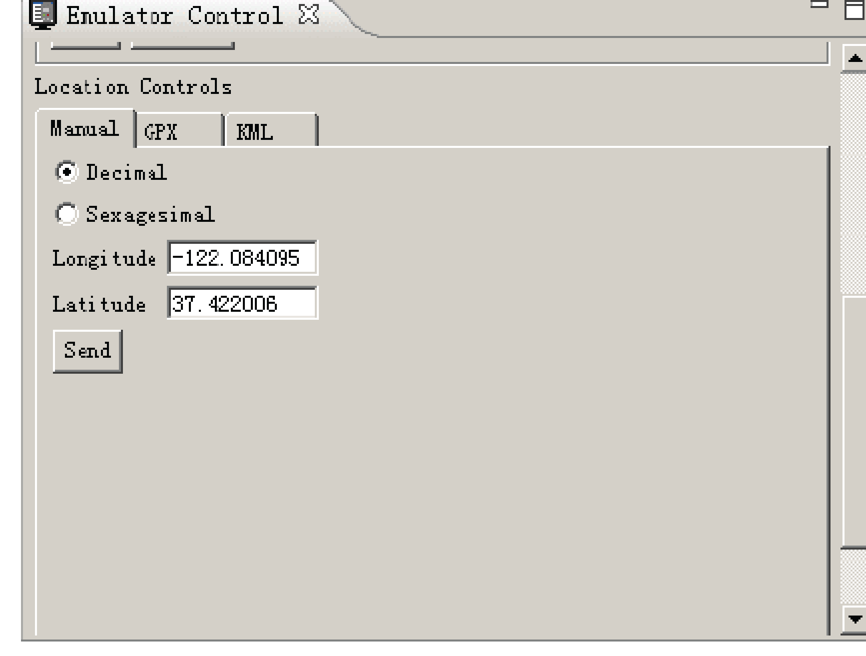


图 4.14　模拟发送 GPS 信号

（5）在模拟器中打开 Maps 应用程序，单击 Menu 按钮，选择 MyLocation。这时程序接收模拟的 GPS 信息并定位，显示如图 4.15 所示。

这里需要注意的是，在新建模拟器时需要选择 Google APIs（Google Inc.）——API Level 8，否则无法支持 GPS 功能，新建模拟器时选择如图 4.16 所示。

4.1.5　使用日志

日志是开发人员在调试程序时必不可少的一个工具，我们可以通过它查看程序的信息，出现异常的情况，以及错误发生的具体代码段等。使用 Logcat 需要以下几个步骤：

（1）选中你需要调试的程序，使其高亮。

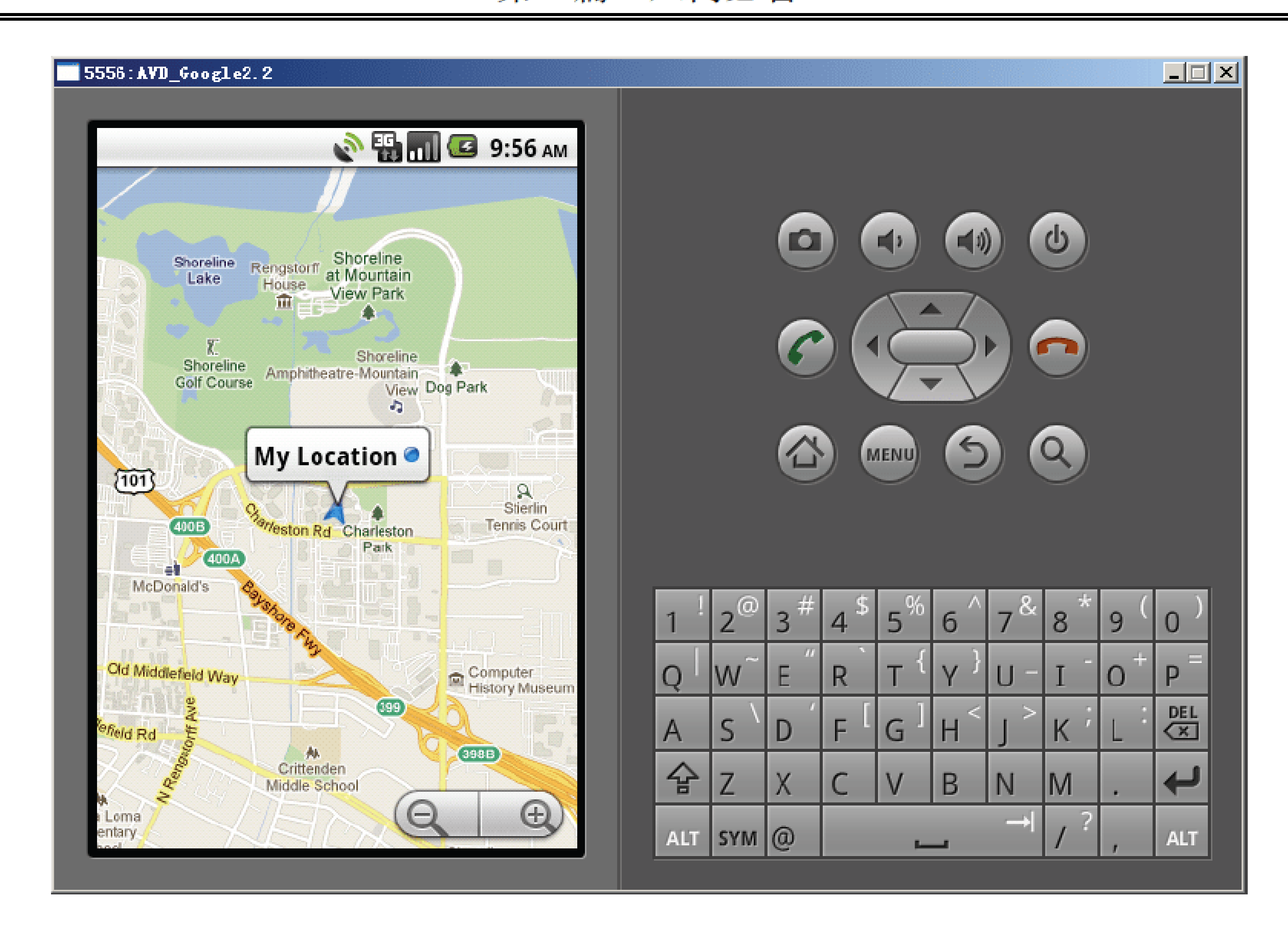


图 4.15　在程序中接收模拟信号

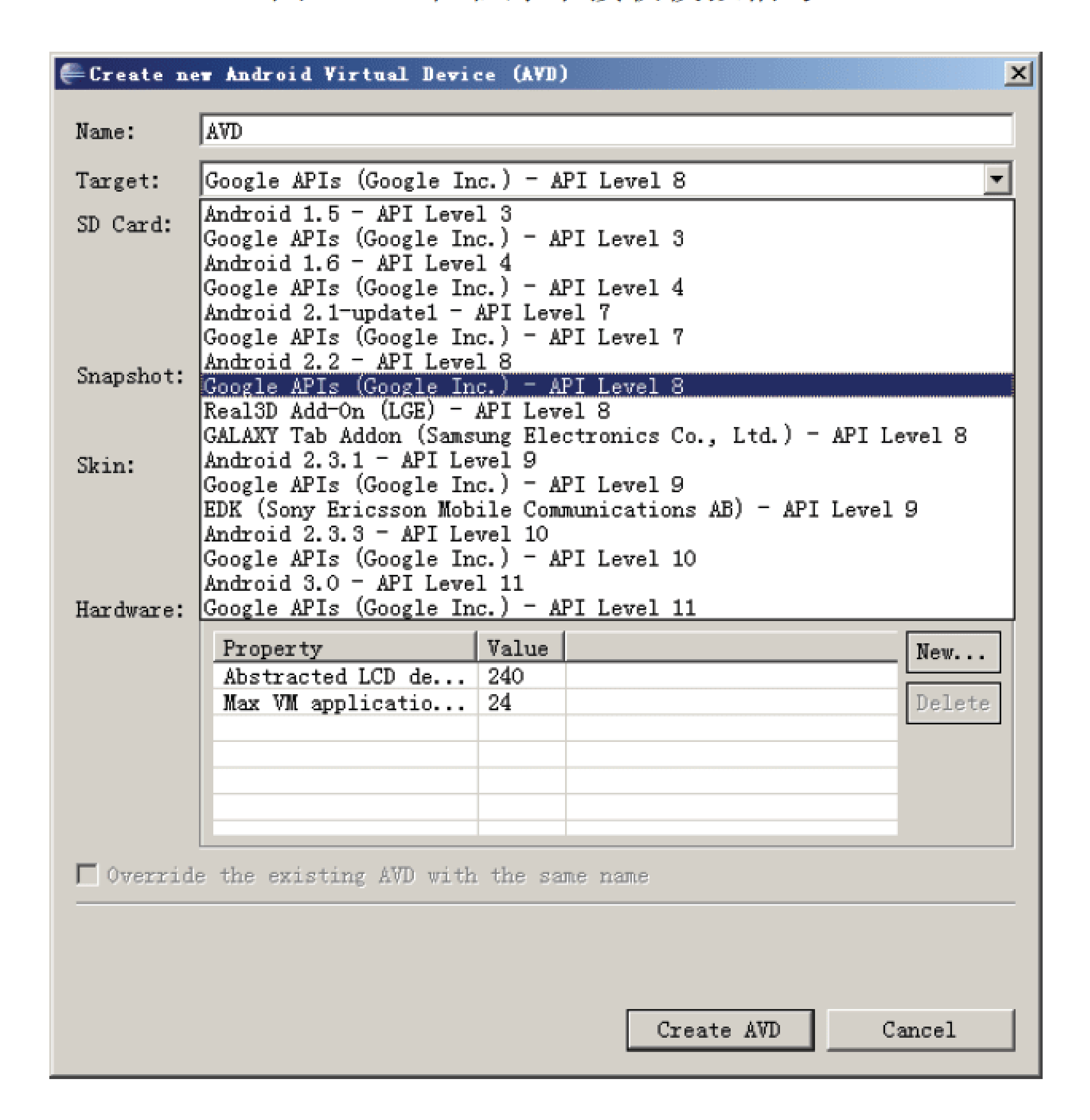


图 4.16　Target 为 Google APIs 的模拟器

（2）在 Windows 菜单中选择 Show View，在弹出的子菜单中选择 Logcat，如图 4.17 所示。

（3）单击后即出现程序的日志输出，显示如图 4.18 所示。

在日志标签页上方有 5 个按钮分别是 V、D、I、W、E。它们的意义是：

（1）V：Verbose，详细信息，即显示所有信息。

（2）D：Dubug，调试过滤器，只输出 D、I、W、E 4 种信息。

（3）I：Information，信息过滤器，只输出 I、W、E 3 种信息。

（4）W：Warning，警告过滤器，只输出 W、E 两种信息。

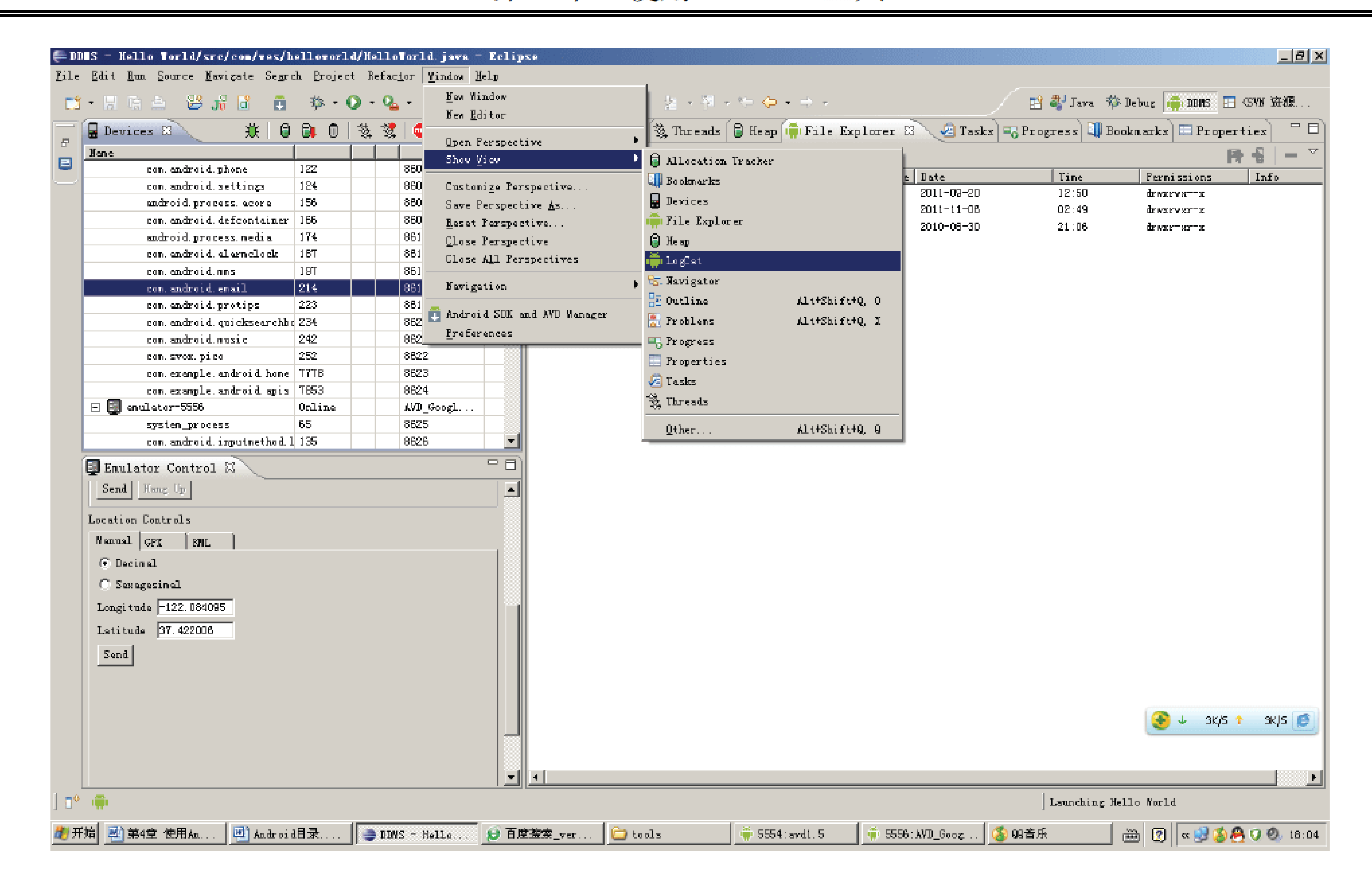


图 4.17　选择显示 Logcat

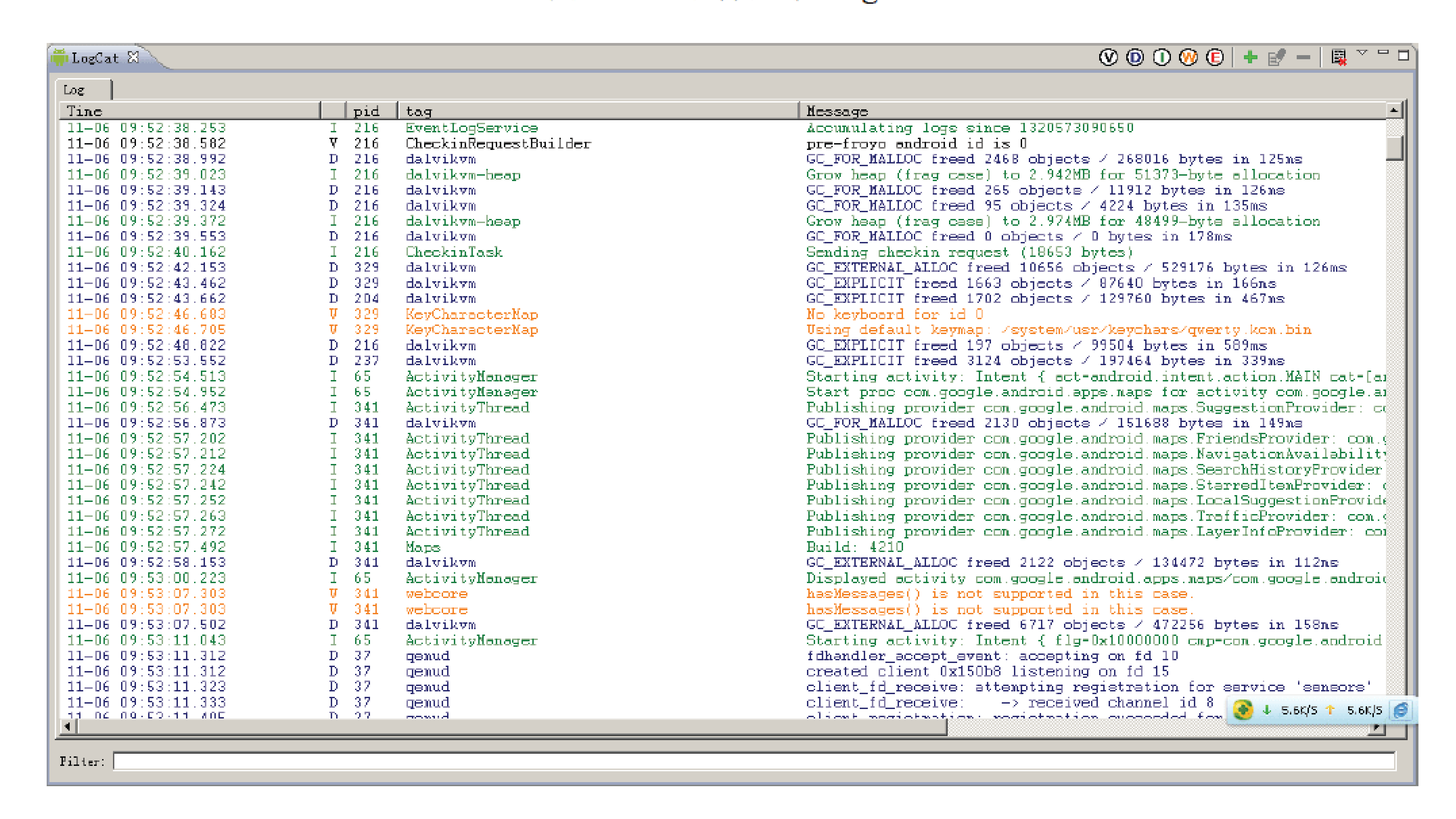


图 4.18　日志显示

（5）E：Error，错误过滤器，只输出 E 一种信息。

它们之间的关系就好比是秘密、机密、绝密 3 种密级，相对身份的人才能知道相对等级的信息。例如，Debug 信息只在开发时可见，用户使用时是看不到该信息的。为了更好地帮助调试，在代码中我们可以添加一些适当的日志输出。例如，我们可以对 HelloWorld 代码进行如下修改：

```
public class HelloWorld extends Activity {
    public static final String TAG = "MY_DEBUG";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
        Log.d(TAG,"第一条日志打印");
    }
```

再次运行程序，我们会发现在日志中多了一条信息，如图 4.19 所示。

图 4.19　第一条日志打印

让我们来观察一下关键代码 Log.d(TAG,“第一条日志打印”)，本行代码的意思是使用 Log 工具打印日志，等级为 D 也就是 Debug 级。其中的两个参数，第一个参数意义为 TAG——标签，第二个参数是希望输出的信息。该行代码就好比是 Java 开发中的。

```
System.out.print();
```

不同的是 Java 在控制台输出，而 Android 在日志中输出。让我们举一反三，既然 Debug 信息是 Log.d()，那么必然还有 Log.v()，Log.i()，Log.w()和 Log.e()。那么让我们实验一下，继续修改 HelloWorld 代码：

```
public class HelloWorld extends Activity {
    public static final String TAG = "MY TAG";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.v(TAG,"Verbose");
        Log.d(TAG,"Debug");
        Log.i(TAG,"Information");
        Log.w(TAG,"Warning");
        Log.e(TAG,"Error");
    }
}
```

运行以上代码，我们可以在 Logcat 中发现如下输出，如图 4.20 所示。

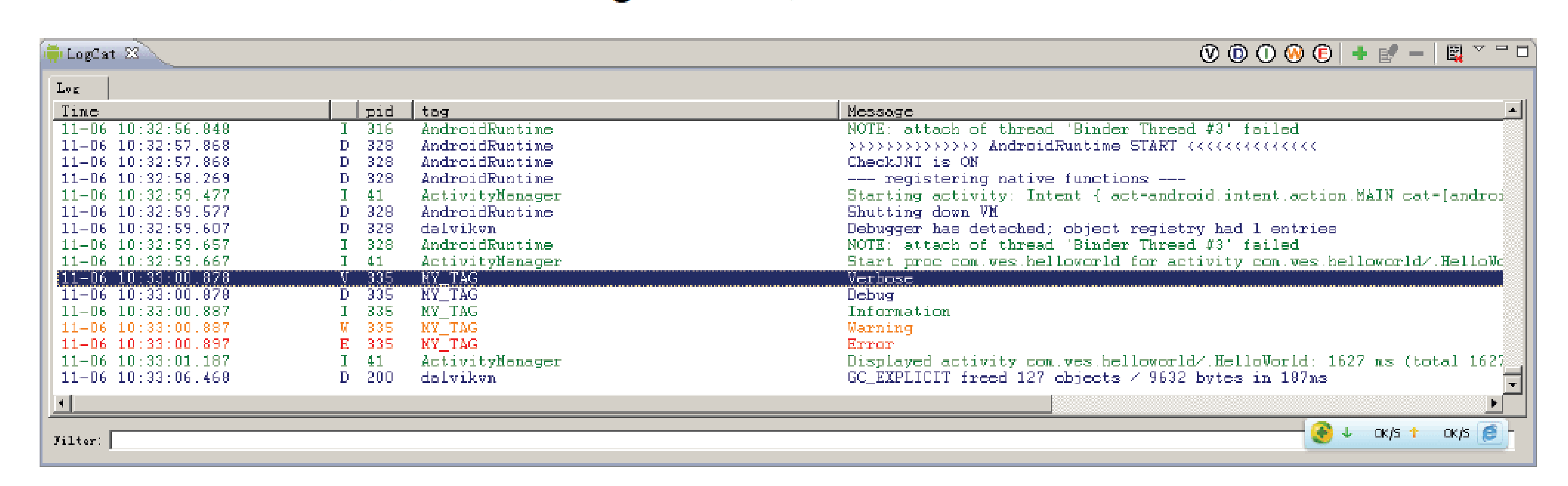


图 4.20　5 种打印输出

为了区分它们之间的区别，每种打印等级的颜色都不一样。从 Verbose 到 Error 其颜色

分别是黑色、蓝色、绿色、橙色、红色。也许读者会说我只想看我添加的调试信息怎么办？这个时候我们可以使用自定义的日志过滤器，使用方法为：单击窗口右上角的+号，在弹出的如图 4.21 所示的对话框中填入相关信息。例如，按照 Tag 过滤，按照进程 Id 过滤，按照日志等级过滤等等。这里我们选择按照 Tag 过滤，单击 OK 按钮后，我们可以看到日志输出中只有我们自己的日志打印了，如图 4.22 所示。

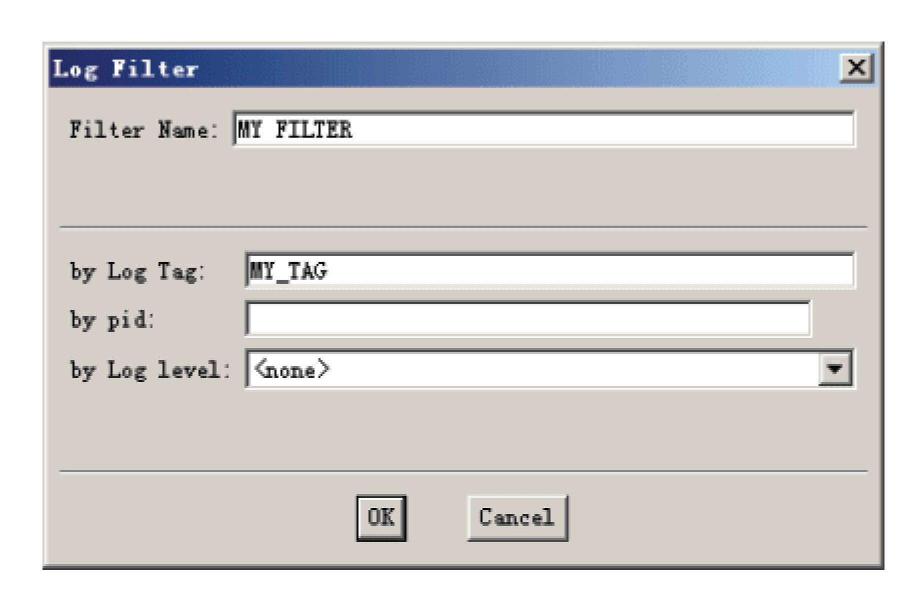


图 4.21　日志过滤器

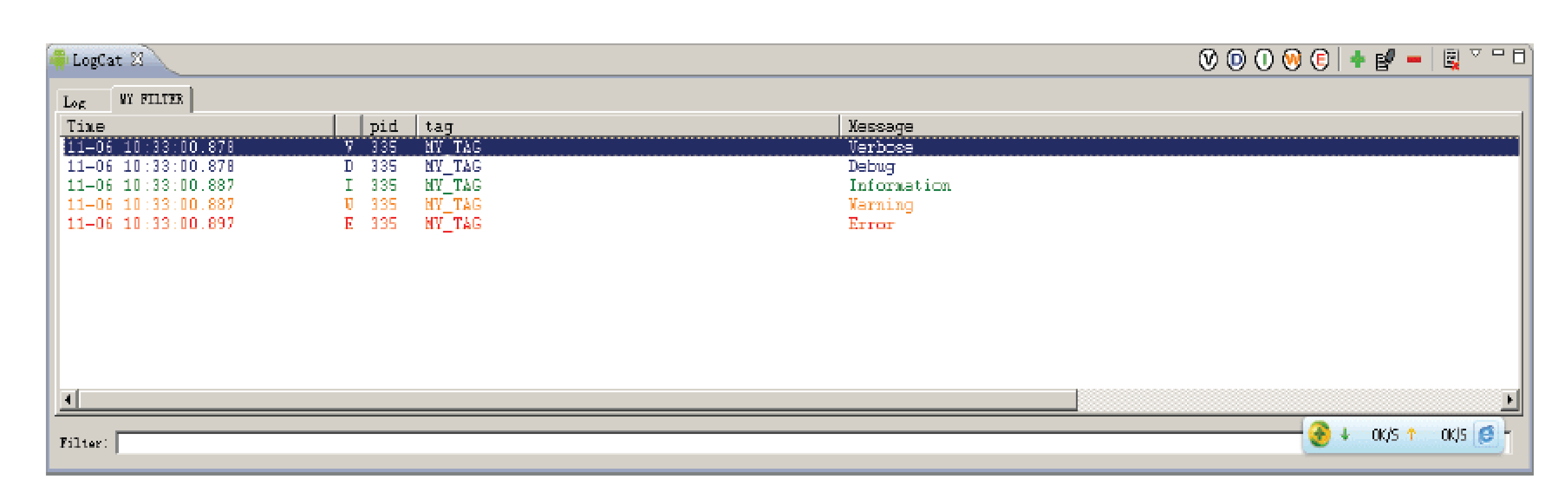


图 4.22　自定义的日志过滤器

4.1.6　使用 Screen Capture 捕捉设备屏幕

截屏对于开发者来说一直是一个麻烦的问题，而 DDMS 的 Screen Capture 功能帮助我们快速方便地截取手机的屏幕。使用方法为：

（1）选择你要截取的设备。

（2）单击 DDMS 左侧面板中上方的相机图标，这时出现截屏窗口，如图 4.23 所示。

（3）单击 Refresh 按钮可以重新获得屏幕画面。

（4）单击 Rotate 按钮可以旋转屏幕，如图 4.24 所示。

（5）单击 Sava 按钮可以保存画面至目标地址。

（6）单击 Copy 按钮可以复制画面，粘贴到需要的地方。

（7）单击 Done 按钮退出截屏。



图 4.23　截屏窗口



图 4.24　旋转屏幕

4.2　使用 Android 调试桥

Android 调试桥（Android Debug Bridge，ADB）是 Android SDK 的一个重要组成部分。它通过命令行直接与设备打交道。在 4.1 节中学习的 DDMS 就有很多功能依赖于 ADB。本节中我们将学习使用 ADB 直接操作设备，而不通过 DDMS 等次级工具。

4.2.1　使用 ADB

要使用 ADB 我们首先要进入 Windows 的命令行窗口，方法为单击“开始”菜单中的“运行”命令，在“运行”对话框中输入 cmd，之后单击“确定”就可以了，如图 4.25 所示。

在弹出的命令行窗口中输入 adb-help，如果出现如图 4.26 所示的界面表示 Android 系统环境变量安装成功，否则请参照第 2 章确认 Android 系统开发环境是否搭建正确。

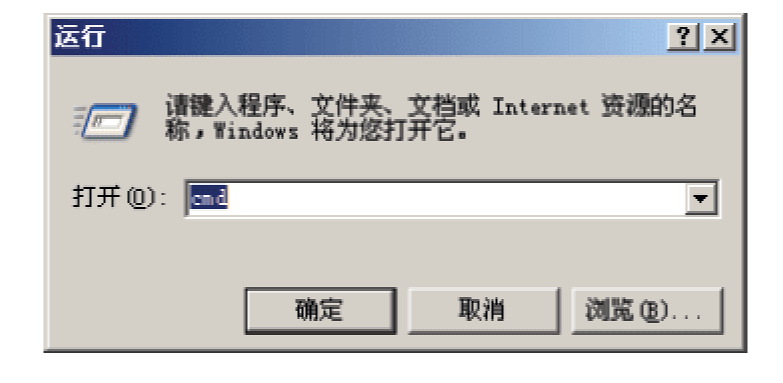


图 4.25　进入命令行窗口

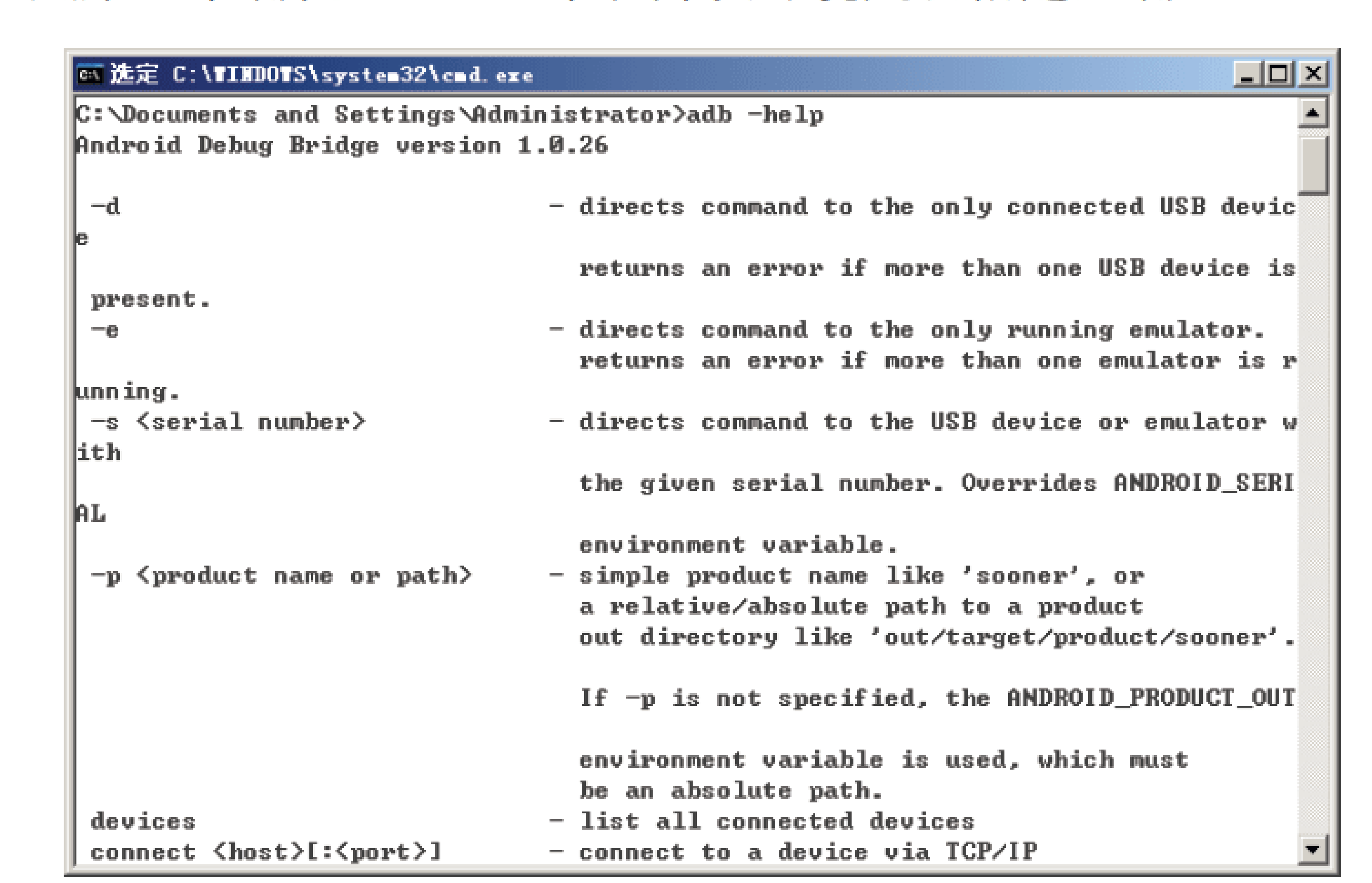


图 4.26　Adb 调试界面

4.2.2　显示连接到计算机的设备

使用 ADB 工具可以很方便地查看所有连接到计算机的设备，只需在命令行中输入：

```
adb devices
```

该命令会列出所有连接到计算机的模拟器和真机的序列号和状态。例如，这里的计算机此刻正在运行一台 HTC Desire，在命令行中输入 adb devices 命令后显示如图 4.27 所示。

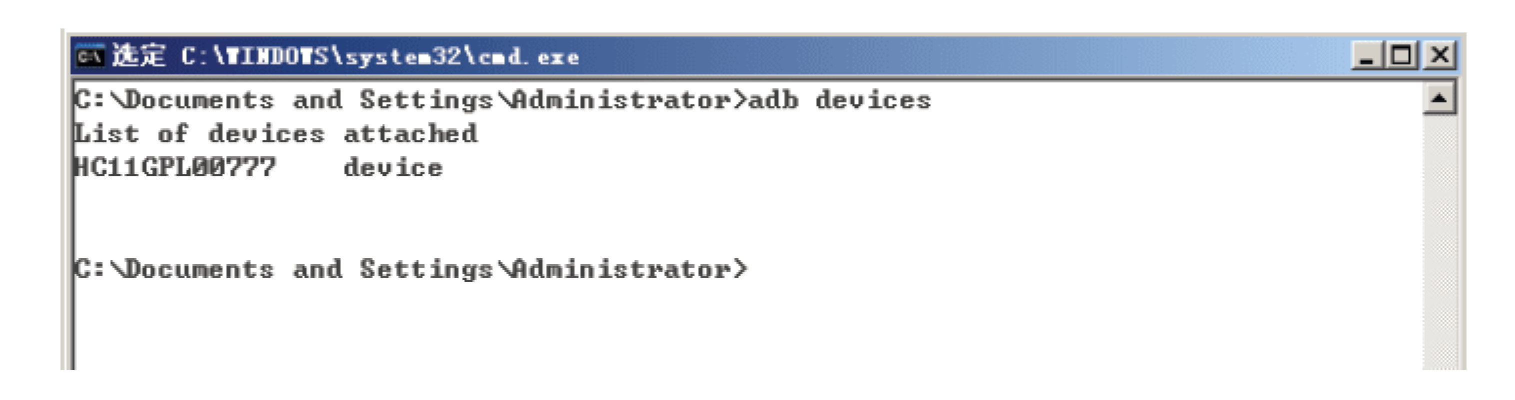


图 4.27　列出所有的连接设备

4.2.3　针对特定设备操作

通过 4.2.2 小节我们得到了设备的序列号，由此序列号我们就相当于得到了设备的名字，通过这个名字我们就可以针对特定的设备发布命令了。命令格式为：

```
adb -s <序列号> 针对该设备的命令
```

例如，获得模拟器的状态：

```
adb -s emulator-5554 get-state
```

这时显示如图 4.28 所示：

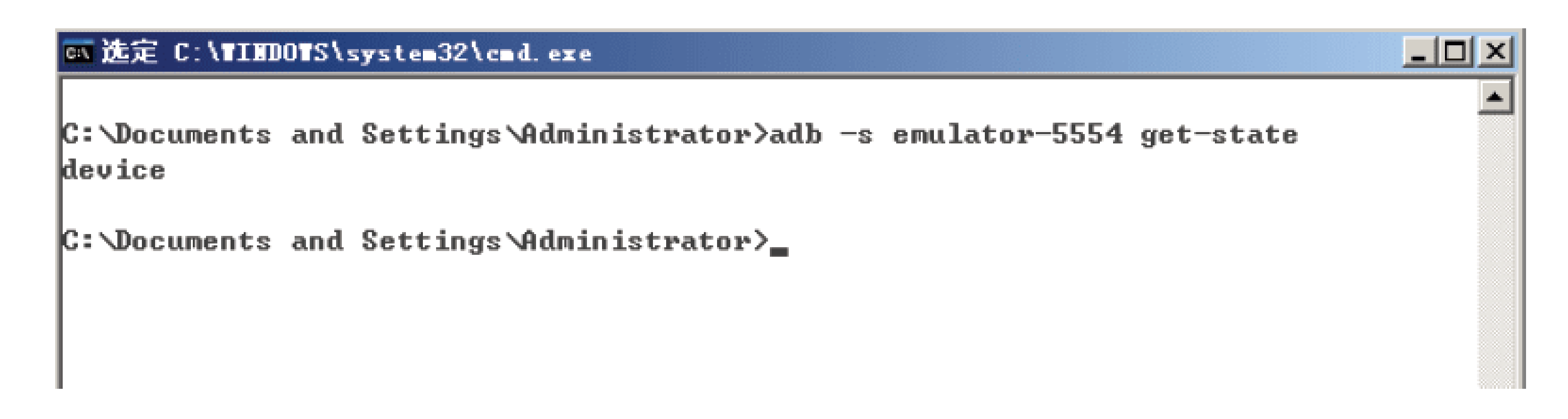


图 4.28　得到特定设备状态

如果你的计算机仅仅连接了一台设备，那么可以使用-d 参数来直接进行操作。例如，当计算机只在运行模拟器时就可以使用如下命令：

```
adb -d get-state
```

这时命令行显示应该如图 4.29 所示。

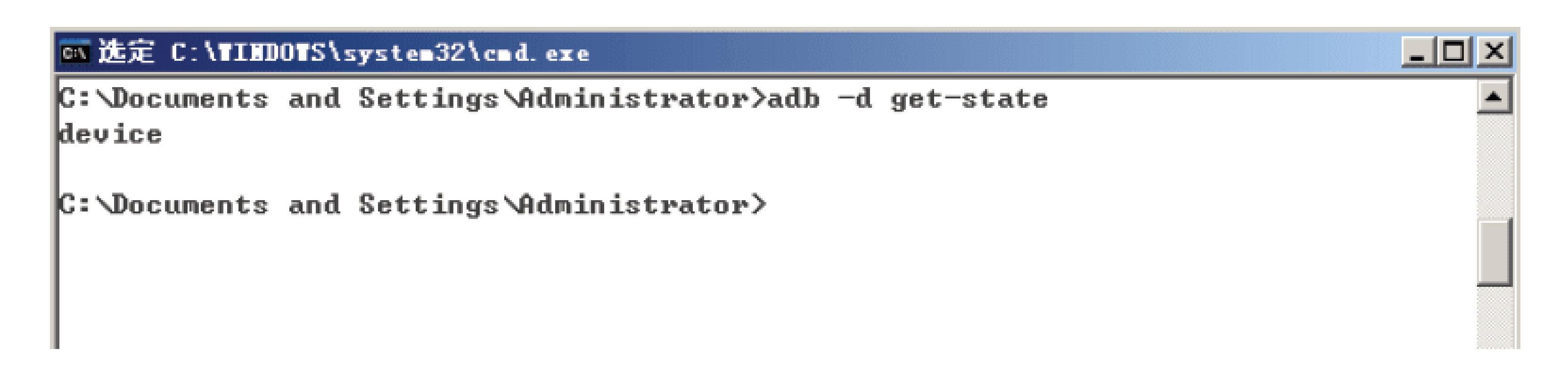


图 4.29　直接操作设备

4.2.4　启动和停止 ADB

在一些情况下，日志记录 Logcat 会无法正常工作，如过于频繁地对调试器进行连接，断开工作。在这些时候，往往无法调试项目，我们就需要重新启动 ADB 服务。

1. 停止ADB服务

停止服务的命令为：

```
adb -s emulator-5554 kill-server
```

这个时候在 Eclipse 的控制台会出现如图 4.30 所示的消息，则表示 ADB 服务停止了。

```
Console
DDMS
[2011-11-06 19:57:37 - DeviceMonitor]Adb connection Error:远程主机强迫关闭了一个现有的连接。
[2011-11-06 19:57:38 - DeviceMonitor]Connection attempts: 1
[2011-11-06 19:57:40 - DeviceMonitor]Connection attempts: 2
[2011-11-06 19:57:43 - DeviceMonitor]Connection attempts: 3
[2011-11-06 19:57:44 - DeviceMonitor]Connection attempts: 4
[2011-11-06 19:57:46 - DeviceMonitor]Connection attempts: 5
[2011-11-06 19:57:48 - DeviceMonitor]Connection attempts: 6
[2011-11-06 19:57:50 - DeviceMonitor]Connection attempts: 7
```

图 4.30　停止 ADB 服务

2. 开始ADB服务

开始服务的命令为：

```
adb -s emulator-5554 start-server
```

开始服务之后，图 4.30 中的尝试连接的消息就不再弹出，表示已经正常连接了，如图 4.31 所示。

```
选定 C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator>adb -s emulator-5554 kill-server

C:\Documents and Settings\Administrator>adb -s emulator-5554 start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *

C:\Documents and Settings\Administrator>
```

图 4.31　开始 ADB 服务

4.2.5　使用 ADB 操作文件和 apk

通过 ADB 可以方便地将文件在手机和计算机之间传递，只需执行一行代码就可以了。当然，你需要知道文件的完整路径。

1. 将文件拷贝到手机

例如，要将 C 盘的 123.txt 文件拷贝到手机上的 data/app 文件夹，命令为：

```
Adb push C:\123.txt /data /app/123.txt
```

显示如图 4.32 所示。

```
选定 C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator>adb push C:\123.txt /data/app/123.txt
62 KB/s (2006 bytes in 0.031s)

C:\Documents and Settings\Administrator>
```

图 4.32　将文件拷贝到手机

2. 将文件从手机拷贝到计算机

例如，要将/data/app/abc.txt 拷贝到 C 盘时，命令为：

```
Adb pull  /data/app/abc.txt C:\abc.txt
```

这个时候显示如图 4.33 所示。

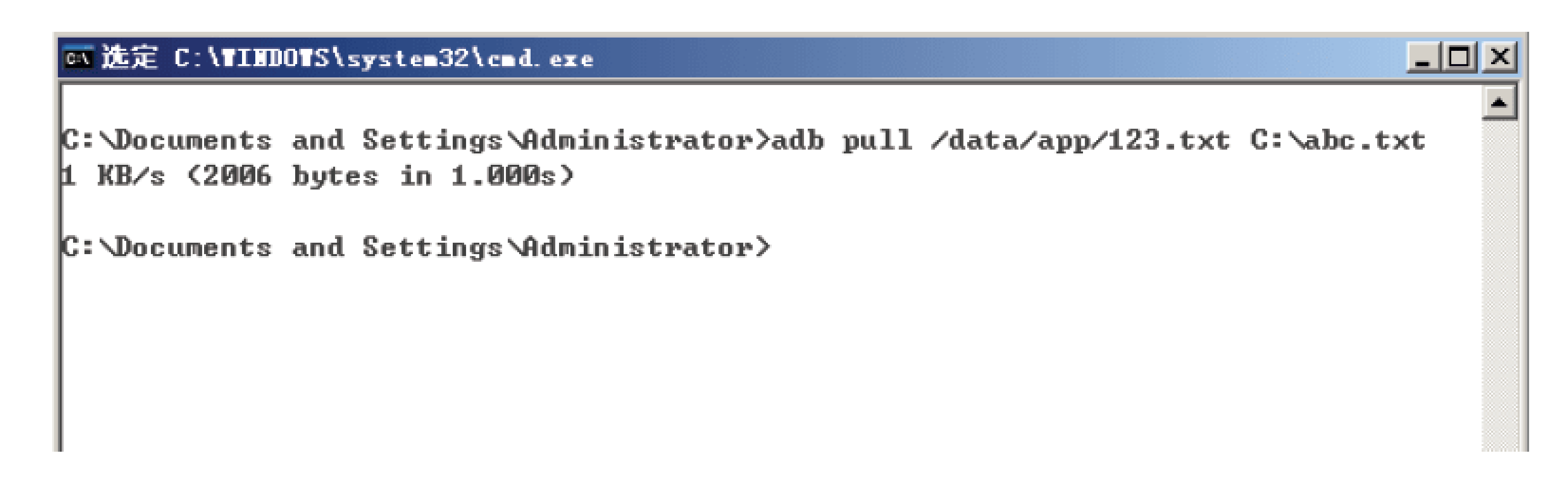


图 4.33　将文件从手机拷贝到计算机

3. 使用ADB安装应用

例如，要将 D:\android\HelloWorld.apk 文件安装到模拟器上，这时要输入的命令为：

```
Adb install D: \android\HelloWorld.apk
```

显示如图 4.34 所示。

```
选定 C:\WINDOWS\system32\cmd.exe
can't find 'D:\android\HelloWorld.apk' to install

C:\Documents and Settings\Administrator>adb install D:\android\HelloWorld.apk
101 KB/s (14645 bytes in 0.140s)
        pkg: /data/local/tmp/HelloWorld.apk
Success

C:\Documents and Settings\Administrator>
```

图 4.34　安装 apk

图中显示安装失败的原因是该 apk 已经存在。

4．重新安装apk

如果对 HelloWorld 不满意可以重新安装，其命令为：

```
Adb install -r D:\android\HelloWorld.apk
```

这时显示如图 4.35 所示。

```
选定 C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator>adb install -r D:\android\HelloWorld.apk

101 KB/s (14645 bytes in 0.140s)
        pkg: /data/local/tmp/HelloWorld.apk
Success

C:\Documents and Settings\Administrator>
```

图 4.35　重新安装

5. 卸载程序

要在手机中卸载程序时要知道程序的完整包名，如卸载 com.wes.helloworld 程序，命

令为：

```
Adb uninstall com.wes.helloworld
```

这时显示如图 4.36 则表示卸载成功。

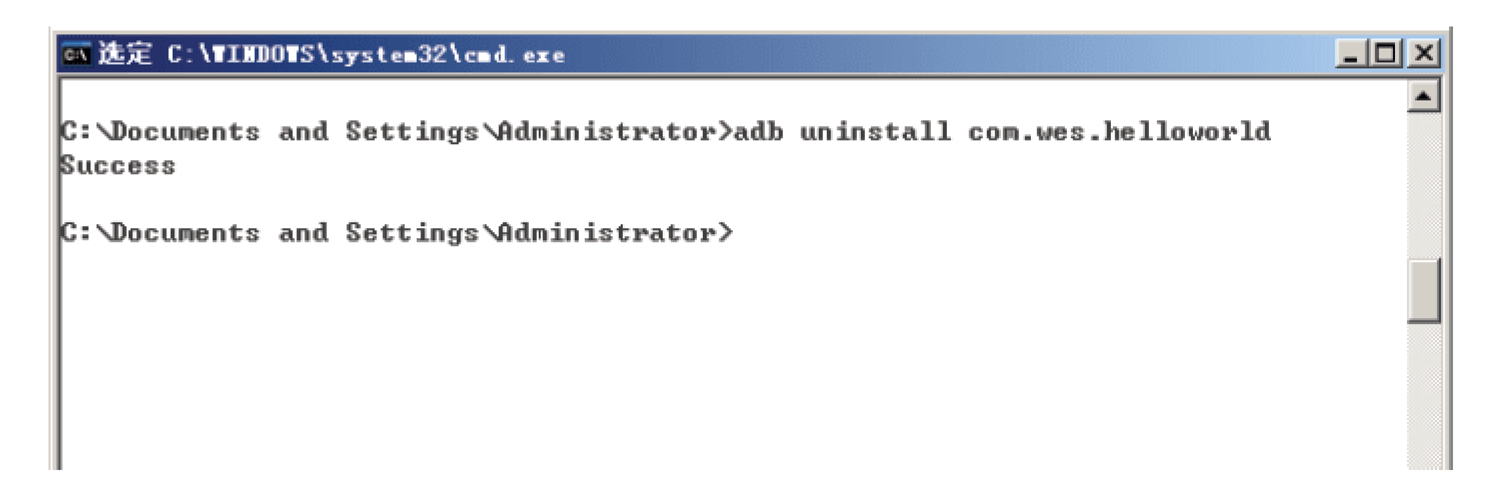


图 4.36　卸载应用

4.2.6　使用 ADB shell

ADB 中包含一个 shell 接口，使用它可以直接操作设备，如查看手机中的所有文件等。下面以查看手机中的所有文件为例，讲解一下如何使用 ADB shell。

首先进入 ADB shell 状态，接着使用 ls 命令得到所有的文件列表，如图 4.37 所示。

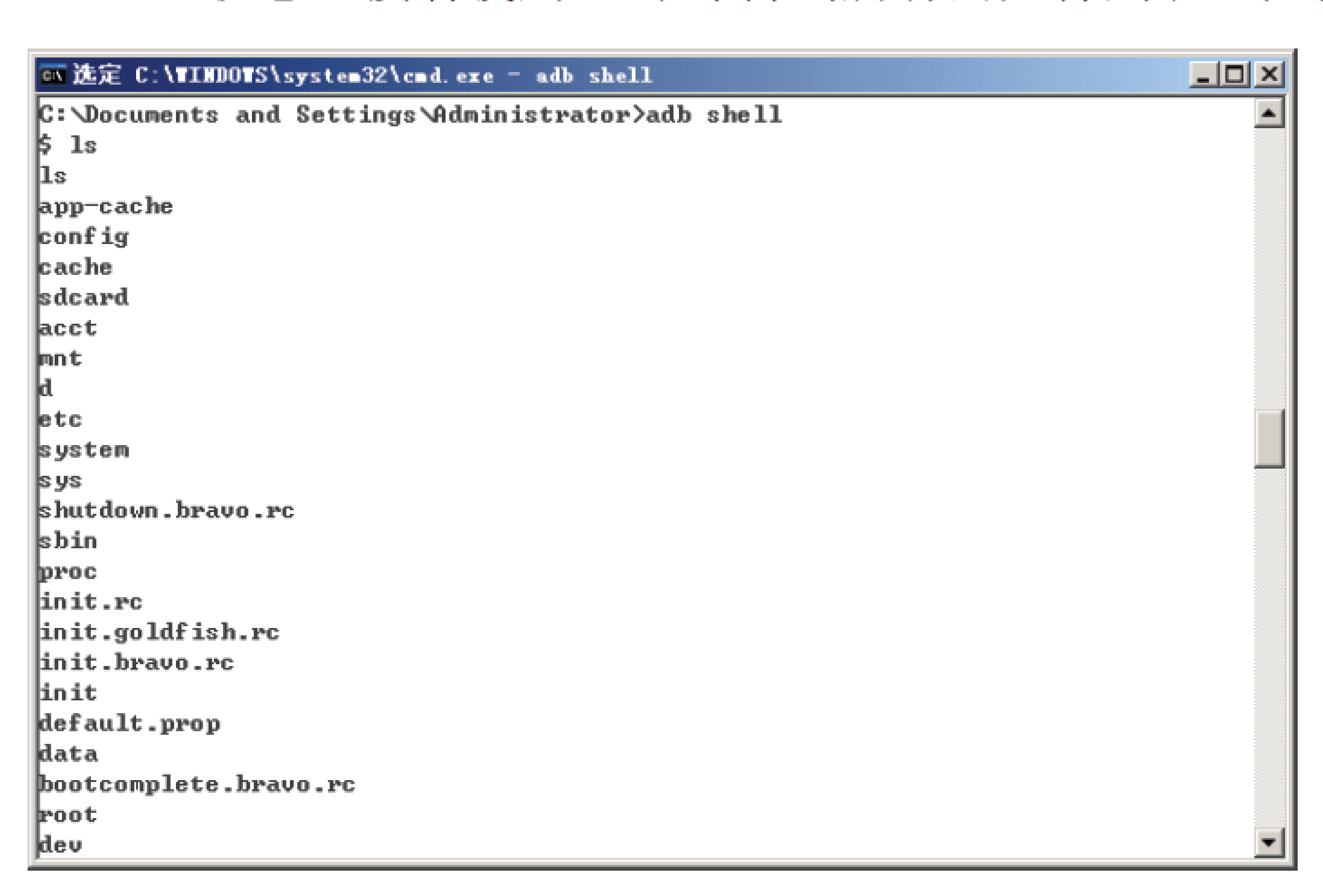


图 4.37　ADB shell 状态

在 ADB shell 状态中更多的命令还需要读者朋友们自行学习 Linux 相关知识，因为 Android 内核是基于 Linux 的，所以在 ADB shell 状态下使用的命令都是 Linux 风格的。操作完毕后输入：

```
exit
```

可以退出 shell 状态。

4.3　使用 AAPT

Android 中的安装文件格式为.apk，如果要在 Android 市场上销售你的程序则必须对程

序进行签名。本节将介绍使用 Android 资源打包工具（Android Assets Packaging Tools，AAPT）对应用程序进行签名的方法。

4.3.1　使用 ADT 导出签名程序

在 Eclipse 中使用 ADT 可以很快速地导出签名程序，其步骤为：

（1）选中要导出的工程，在右键菜单中选择 Android Tools，在弹出的子菜单中选择 Export Signed Apliacation Package，如图 4.38 所示。

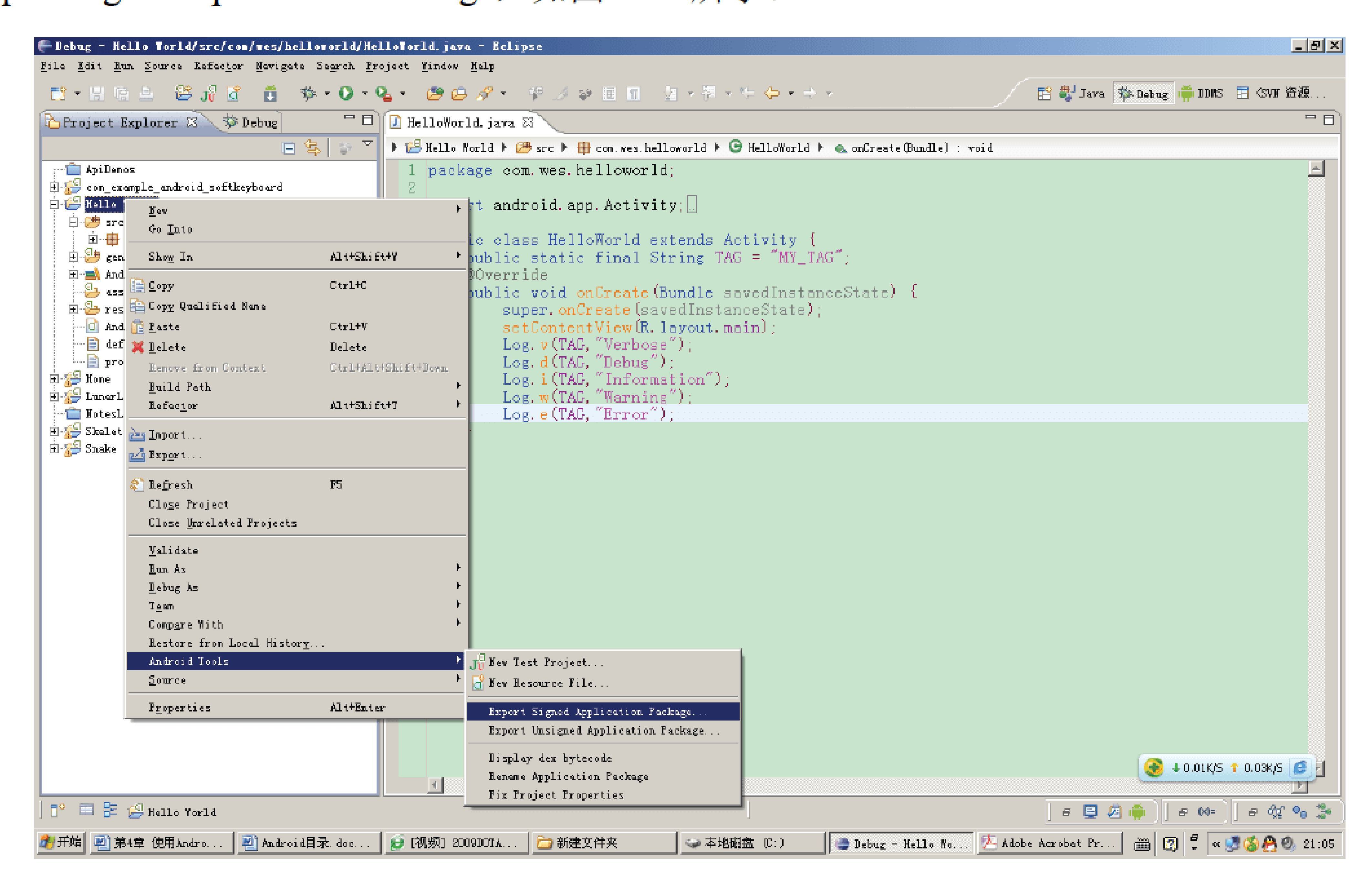


图 4.38　选中 Export Signed Apliacation Package

（2）在弹出的对话框中选择要导出的工程，如果之前已经选择好则不必修改，如图 4.39 所示。

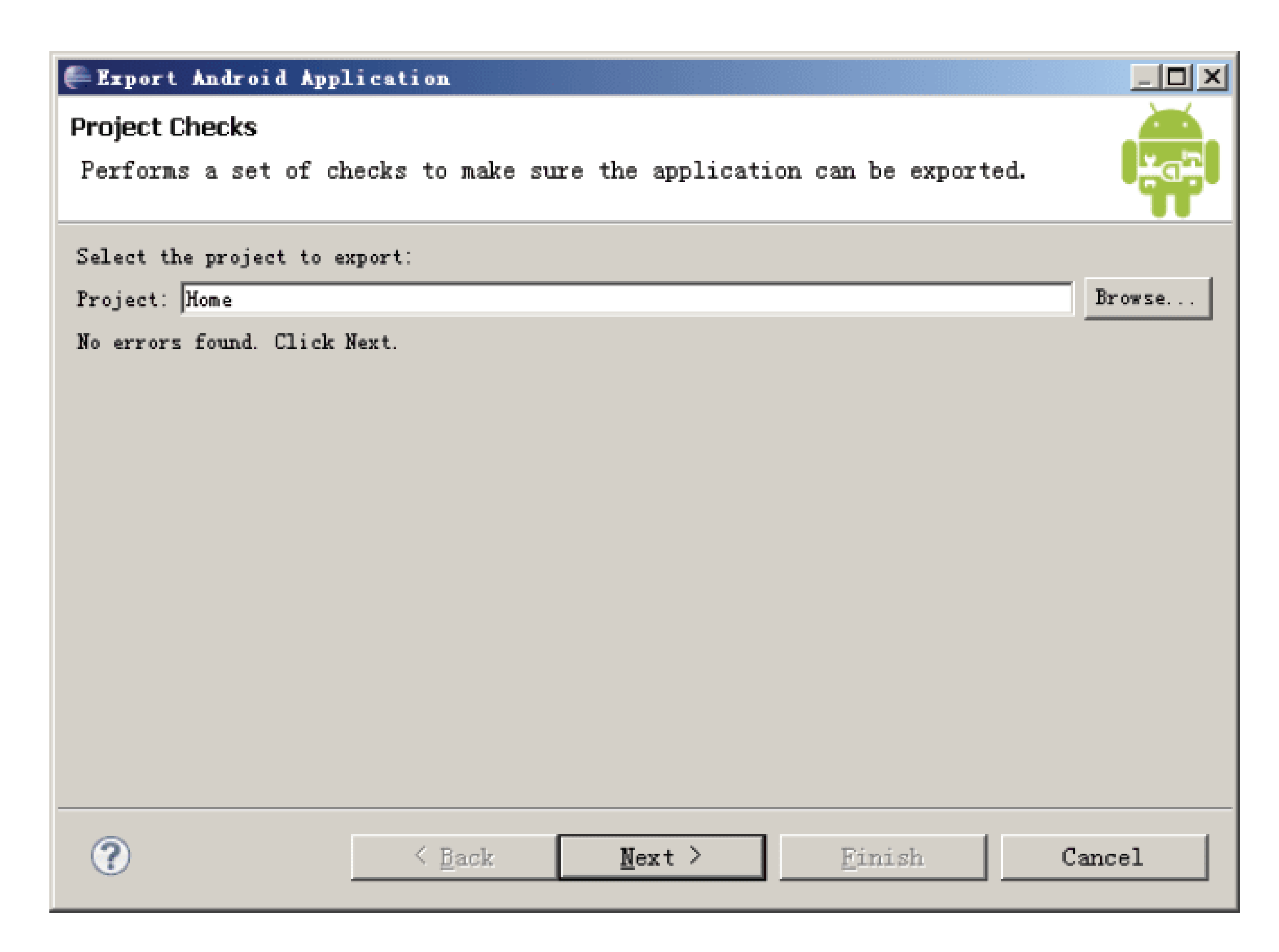


图 4.39　选择导出的工程

（3）单击 Next 按钮，进入 Keystore 设置界面，按照提示输入路径和密码并确认，如图 4.40 所示。

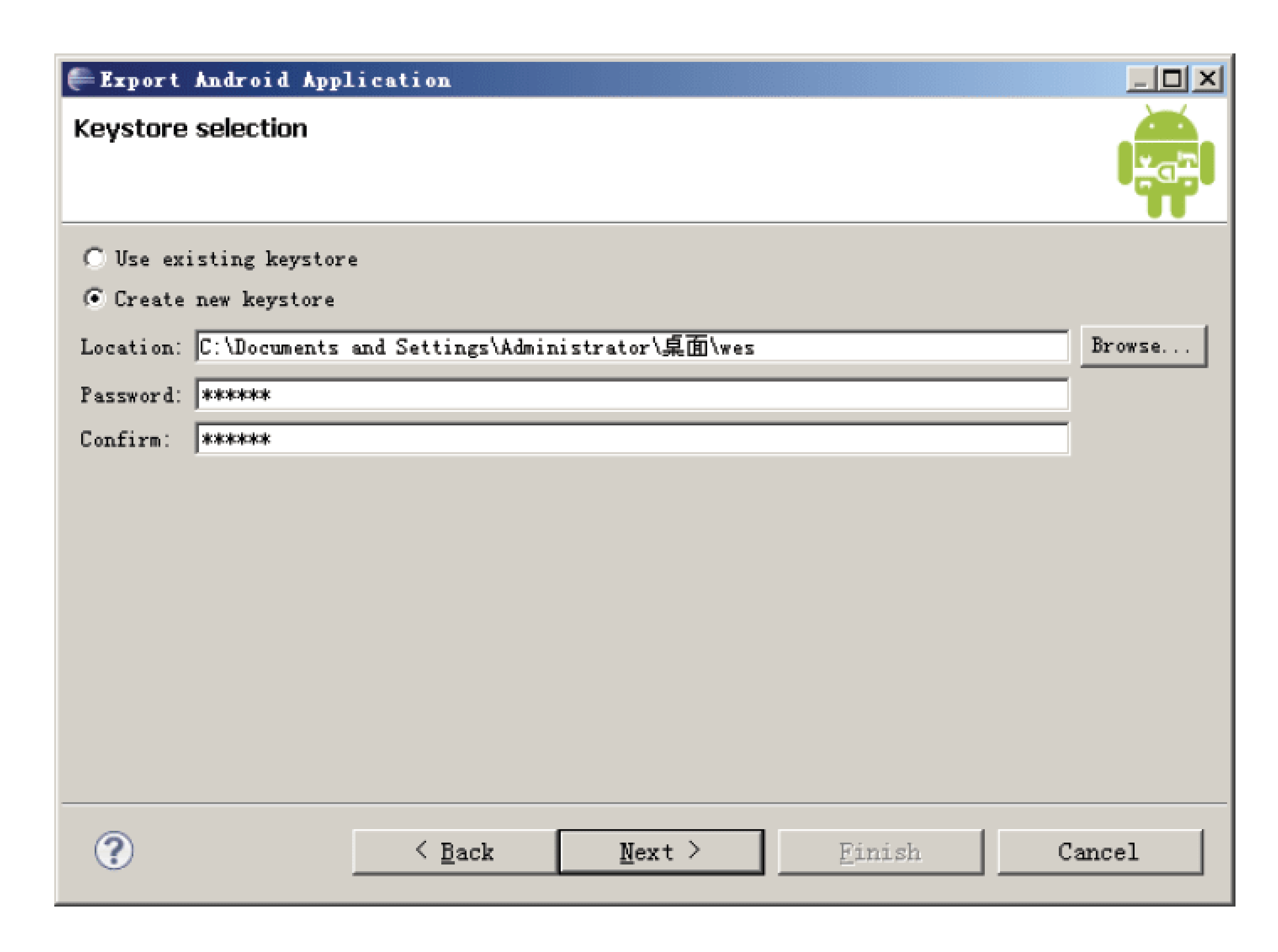


图 4.40　Keystore 设置

（4）单击 Next 按钮，进入如图 4.41 所示 Key 生成界面。按照提示输入 Alias（别名）、Password（密码）、Validity（有效日期）、First and Last Name（作者姓名）等必填项目。

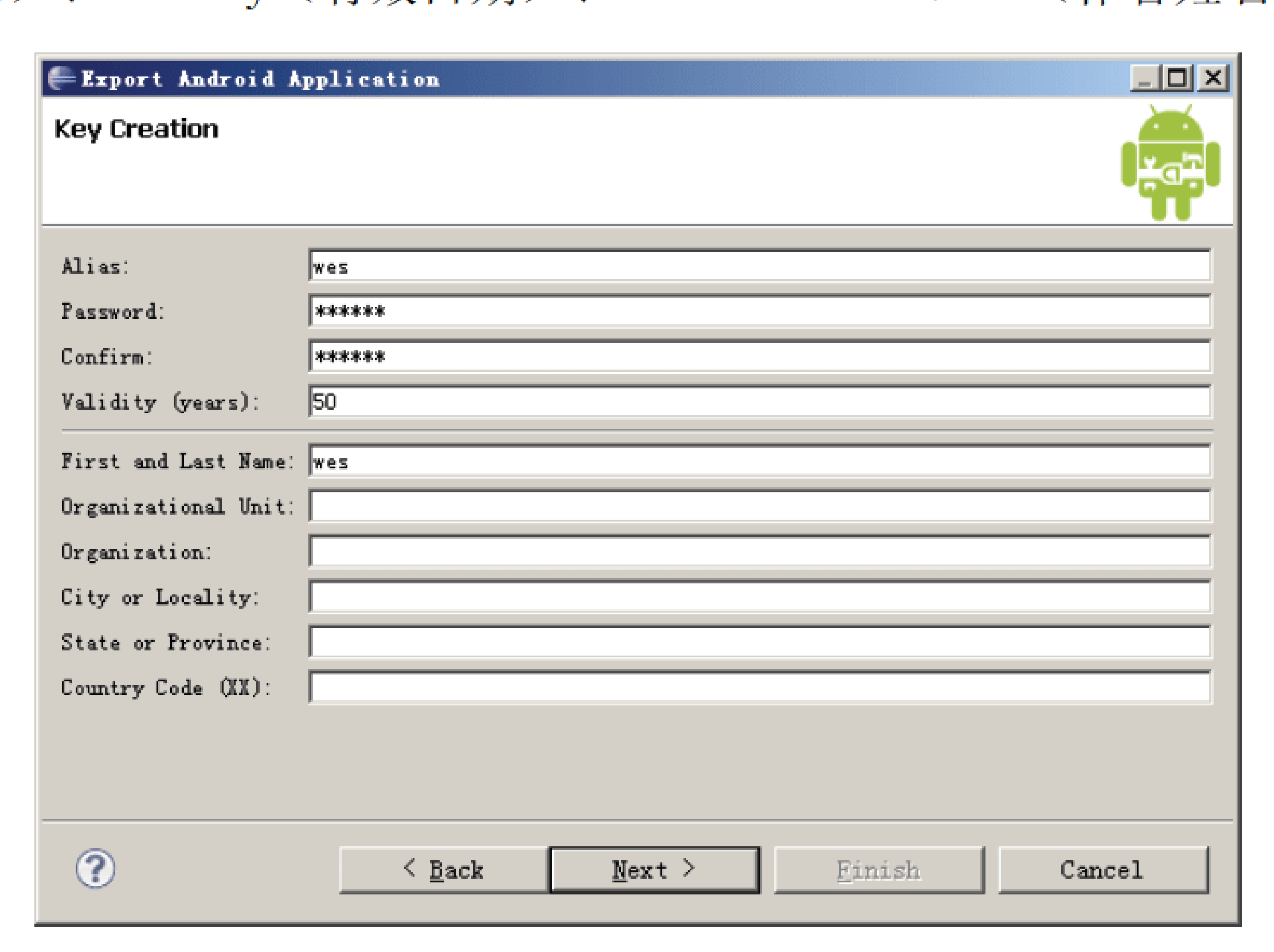


图 4.41　Key 生成界面

（5）单击 Next 按钮，选择 apk 保存的具体路径，显示如图 4.42 所示。

（6）单击 Finish 按钮完成导出，这时在填写的路径下已经有一个签名完毕的 apk 文件了。

4.3.2　使用命令行生成签名 apk 文件

在命令行生成 apk 文件要分为以下两个步骤：

（1）在命令行中利用 Keytool 生成 Keystore 签名文件。

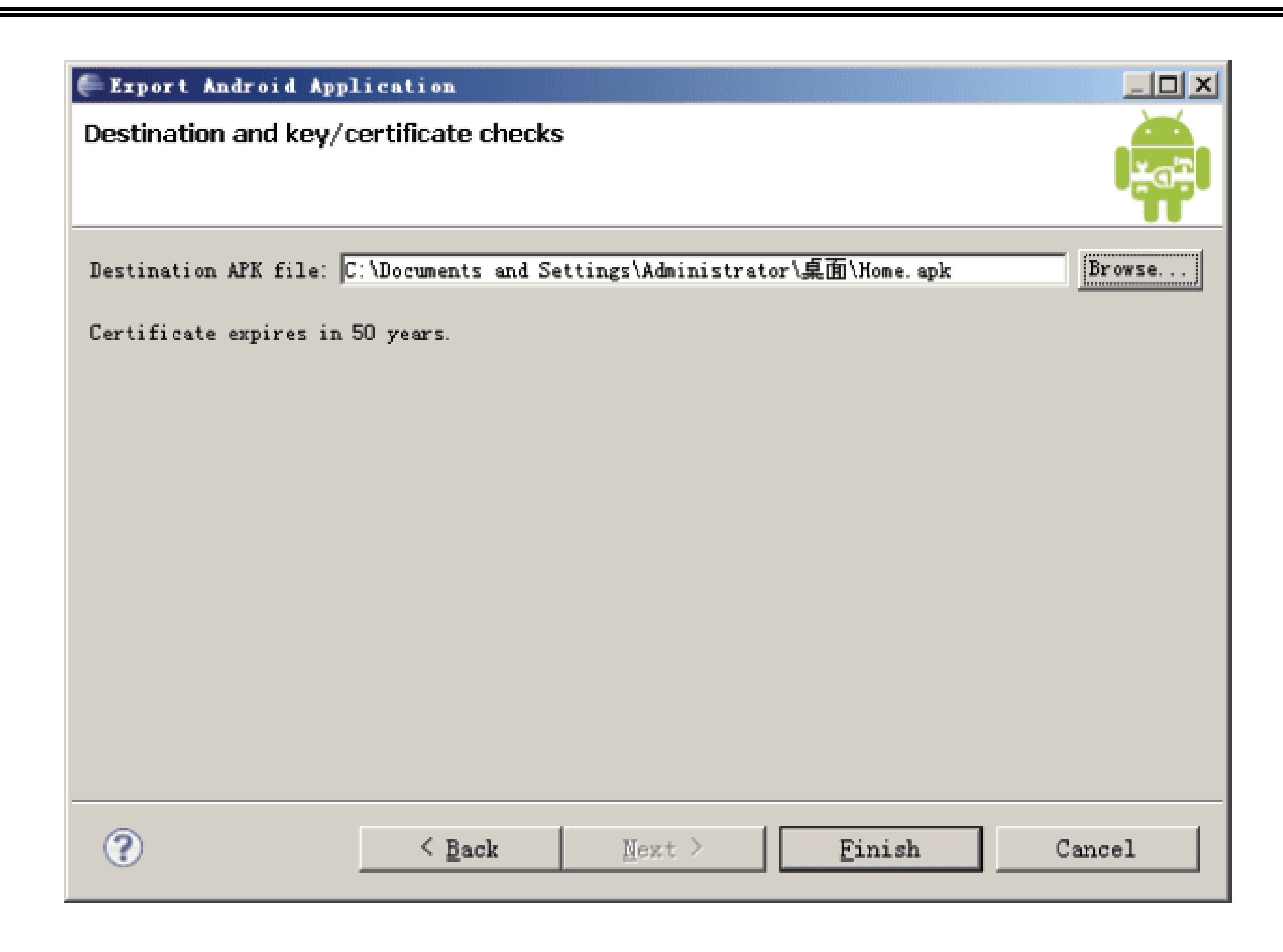


图 4.42　保存 apk 路径

（2）利用 jarsigner 工具替未签名的 apk 签名。

1. 生成Keystor

（1）在命令行中是使用 cd 命令进入你希望存放 Keystore 的位置，如要进入在 D 盘新建的 Keystore 文件夹，命令如下：

```
d:
cd android\keystore
```

这时运行效果如图 4.43 则表示进入了希望的目录。

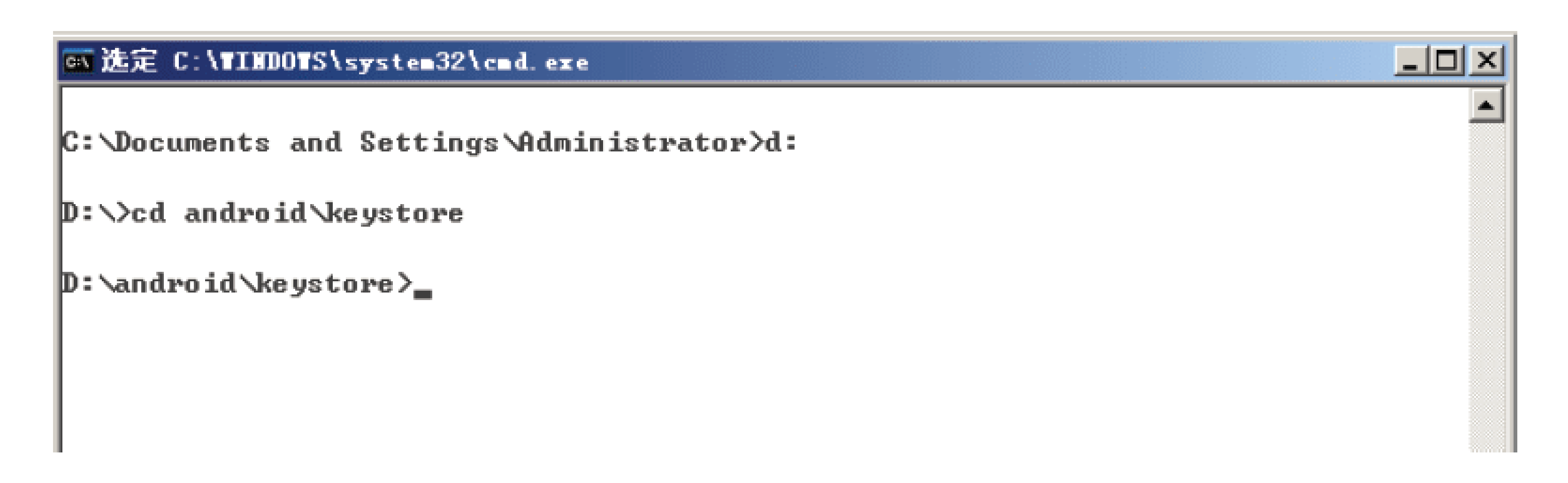


图 4.43　进入目标文件夹

（2）在命令行中输入：

```
keytool -genkey -alias wes_android.keystore -keyalg RSA -validity 50000
-keystore wes_android.keystore
```

这时会提示你输入一些相关的信息，按照要求输入就可以了，如图 4.44 所示。

这里的命令行意义是：

- keytool-genkey：使用 keytool 产生 key。
- -alis　wes_android.keystore ：-别名 wes_android.keystore。
- -keyalg　RSA：-key 加密方式 RSA 方式。
- -validity　50000：-有效日期 50000 天。
- -keystore　wes_android.keystore：-keystore 文件名 wes_android.keystore。

这个时候在目标文件夹中就会生成一个 Keystore 文件了。使用 dir 命令可以查看该目

录下的所有文件，此时显示如图 4.45 所示。

```
选定 C:\WINDOWS\system32\cmd.exe
D:\android\keystore>keytool -genkey -alias wes_android.keystore -keyalg RSA -val
idity 50000 -keystore wes_android.keystore
输入keystore密码:  1234567890
您的名字与姓氏是什么?
  [Unknown]:  wes
您的组织单位名称是什么?
  [Unknown]:  cyber
您的组织名称是什么?
  [Unknown]:  company
您所在的城市或区域名称是什么?
  [Unknown]:  suzhou
您所在的州或省份名称是什么?
  [Unknown]:  jiangsu
该单位的两字母国家代码是什么
  [Unknown]:  CN
CN=wes, OU=cyber, O=company, L=suzhou, ST=jiangsu, C=CN 正确吗?
  [否]:  y

输入<wes_android.keystore>的主密码
        (如果和 keystore 密码相同，按回车):

D:\android\keystore>_
```

图 4.44　生成 Keystore

```
选定 C:\WINDOWS\system32\cmd.exe

D:\android\keystore>dir
 驱动器 D 中的卷没有标签。
 卷的序列号是 44B4-D18D

 D:\android\keystore 的目录

2011-12-28  20:38    <DIR>          .
2011-12-28  20:38    <DIR>          ..
2011-11-07  20:55            14,644 HelloWorld.apk
2011-11-07  21:00            14,551 test.apk
2011-12-28  20:38             1,351 wes_android.keystore
2011-10-31  17:16             1,341 wes_keystore
               4 个文件         31,887 字节
               2 个目录 35,194,298,368 可用字节

D:\android\keystore>_
```

图 4.45　查看 Keystore 是否生成

2．生成签名apk文件

接下来生成真正的签名 apk 文件，需要使用刚才生成的 Keystore 文件。例如要为 HelloWorld.apk 签名，命令如下：

```
jarsigner  -keystore  wes android.keystore  HelloWorld.apk  wes android.
keystore
```

注意这里可以直接使用 jarsigner 命令的原因，是在系统变量中已经将 jdk 的 bin 目录配置到系统 Path 中，所以这里可以直接使用，如果不能正常使用，请确认系统环境是否配置正确，或进入 jarsigner.exe 所在目录后再使用该命令。此时命令行应当显示如图 4.46 所示。

3．验证

最后我们还需要验证一下 apk 是否签名成功，如要验证 HellWorld.apk 是否签名成功，使用命令为：

```
Jarsigner -verify HelloWorld.apk
```



图 4.46　使用 jarsigner 生成签名 apk

此时命令行会提示已验证字样则表示成功了，如图 4.47 所示。



图 4.47　验证 apk

4.4　小　　结

本章我们学习了如何使用 DDMS、ADB、AAPT 等工具。它们的功能都非常强大，如果能利用好这些工具会让我们的开发事半功倍。本章的重点是 DDMS 的使用，因为它是很多强大工具的集合，也是开发中最经常用到的工具。其难点是 ADB 的使用，这涉及到 Linux 的相关知识。下一章我们将进入第 2 篇，真正开始学习 Android 具体代码的编写。

第 2 篇　界面开发

第 5 章　探索界面 UI 元素

本章是 Android 开发中非常重要的一章，因为本章着重介绍 Android 界面开发中常用的一些组件，并通过实例讲解了视图、组件、布局之间的关系，只有学习好本章才可以编写出优秀的界面。希望读者可以多抽出一些时间阅读本章加深理解，并进行实践练习。要知道：界面是否优秀和友好，往往是判断一个程序好坏的必要因素。

5.1　认识 Android 视图、Widget 以及布局

在学习 Android 编程中需要使用的一些常用组件前，先向大家介绍一些基础的理论知识，希望读者能认真阅读，不要操之过急。理解了一些基础知识后再学习组件的使用，会更有利于加深记忆。接下来做一个简单的介绍。

1．视图简介

现在，请读者想象一下：当打开一个程序，第一眼看到的是什么？没错，答案是界面！那么在 Android 中界面是什么？由第 4 章，我们知道一个 Android 程序由一个或者多个 Activity 组成。而界面则显示在 Activity 中，Activity 本身并不现实，也就是说 Activity 只是一个容器。那么一个容器里装的是什么呢？如文本框、按钮、单选项、多选项、编辑框等等。这些组件我们将之统称为 View，中文称之为“视图”。

在 Android SDK 中有一个 android.view 包，该包中是一些界面绘制相关的接口和类。当然，更多的时候我们所说的 View 并不是指这个包而是该包中的一个类——android.view.View。

View 类实质上是屏幕上的一块矩形区域，既然是一个矩形区域那么必定有宽和高。在这里先强调一下：在使用 xml 编辑界面时，一个组件必须有宽和高的属性，否则编译会出错。View 类是所有的 Widget 和布局的基类，在下一节中即将介绍的各类组件都是其子类，其实从名称上就可以看出来，如 TextView（文本框）、EditText（编辑框）、ScrollView（滚动视图）等等。

在图 5.1 中可以看出，几乎所有的元素都是 View。

2．Widget简介

了解了 View 的基本概念后再来看 View 的各个子类。在 Android SDK 中有一个 android.widget 包，在这个包里包含了很多的类。例如，前文提到的 TextView（文本框）、EditText（编辑框）、ScrollView（滚动视图）以及布局（Layout）等。通常这些组件都是继承于 View 类，在下一节“必须了解的 Widget 组件”中将向读者着重介绍这一部分，在

这里就不再赘述了。

在图 5.1 中标注出了一些常见的 Widget，读者可以先做一个初步的了解。

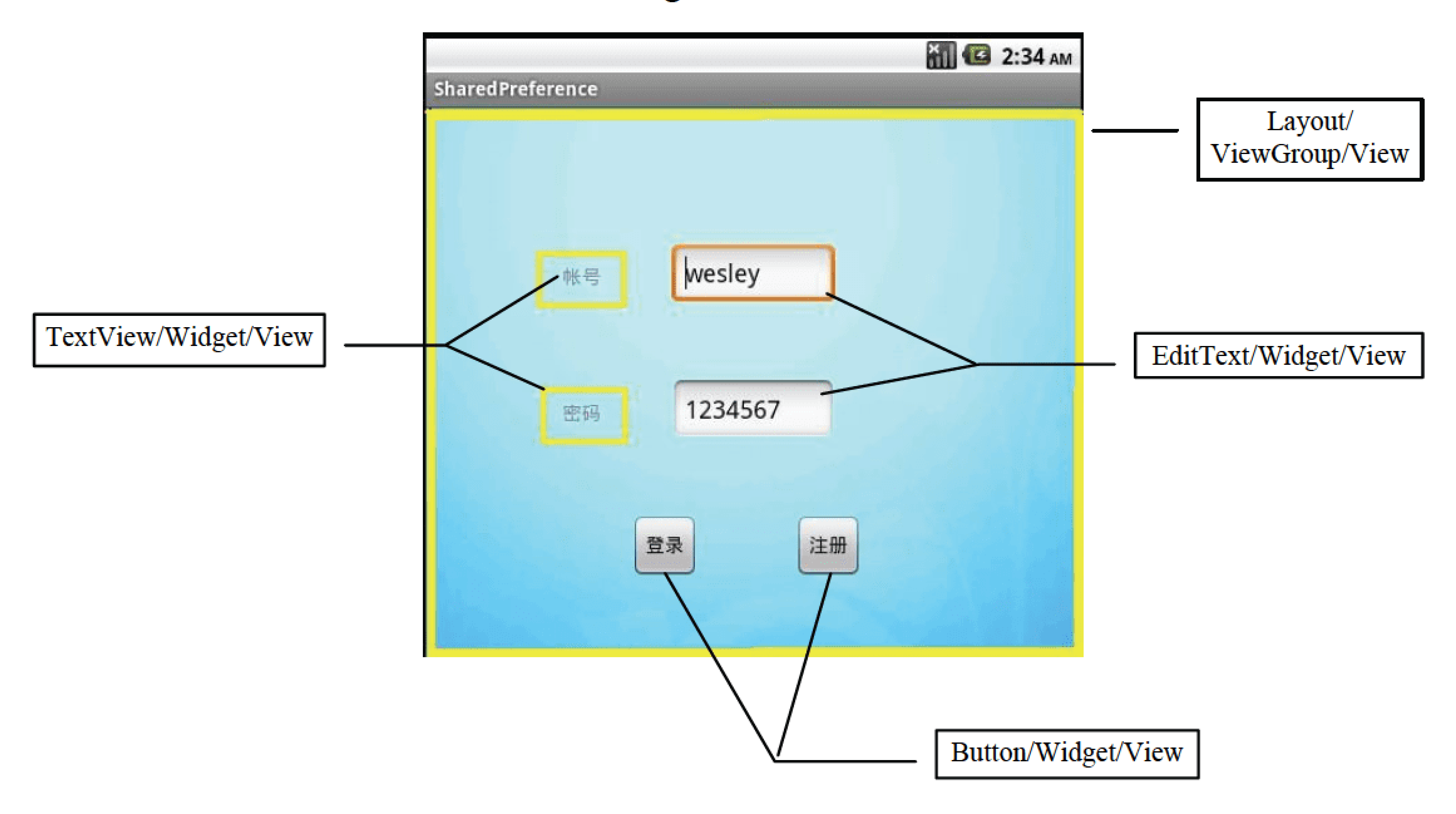


图 5.1　Widget、View、ViewGroup 示例图

3. ViewGroup简介

在前面，我们知道视图需要放在一些容器中显示，容器本身并不显示，如 Activity。事实上，Android SDK 中有一类叫做 android.widget.ViewGroup。顾名思义，这个类是 View 的容器类，也是 View 的子类。一个 ViewGroup 对象负责对添加进去的各个 View 对象进行布局。一个 ViewGroup 对象也可以添加入另一个 ViewGroup 对象，因为一个 ViewGroup 同样继承于 android.view.ViewGroup。

ViewGroup 是一个抽象类，其典型的实现类是布局。一个布局按照一定的规则对添加在其内的子 Widget 进行布局。例如，android.widget.LinearLayout 类，其简单地按照横向或者纵向排列子 Widget；又如 android.widget.AbosoluteLayout 类，其按照绝对位置摆放子 Widget，也就是说可以显示地为每一个子 Widget 指定一个精确的坐标。

总而言之，一个 ViewGroup 可以容纳一些你需要的子 Widget，并按照一定的规则排列。如果一个 ViewGroup 对象不够，那么可以将多个 Widget 嵌套，最终实现设想的界面。

图 5.1 中的整体就是一个 Layout 布局，同时也可以称之为 ViewGroup，当然它更是一个 View。

5.2　必须了解的 Widget 组件

从本节开始将介绍平时开发中常用的一些 Widget 组件，希望读者能仔细阅读本节，并多多练习，因为这在以后的开发过程中会经常出现。闲言少叙，马上开始 5.2.1 小节的讲解吧。

5.2.1　使用可滚动的文本控件——TextView

从本小节开始将向大家介绍一些常用的 Widget 类。首先介绍的是 TextView，它是 Android SDK 中最简单也是最重要的一个类。其用处是向用户简单地显示一些固定的字符串。首先我们来看一个最简单的小程序——HelloWorld。

运行程序，我们会看到如图 5.2 所示界面。

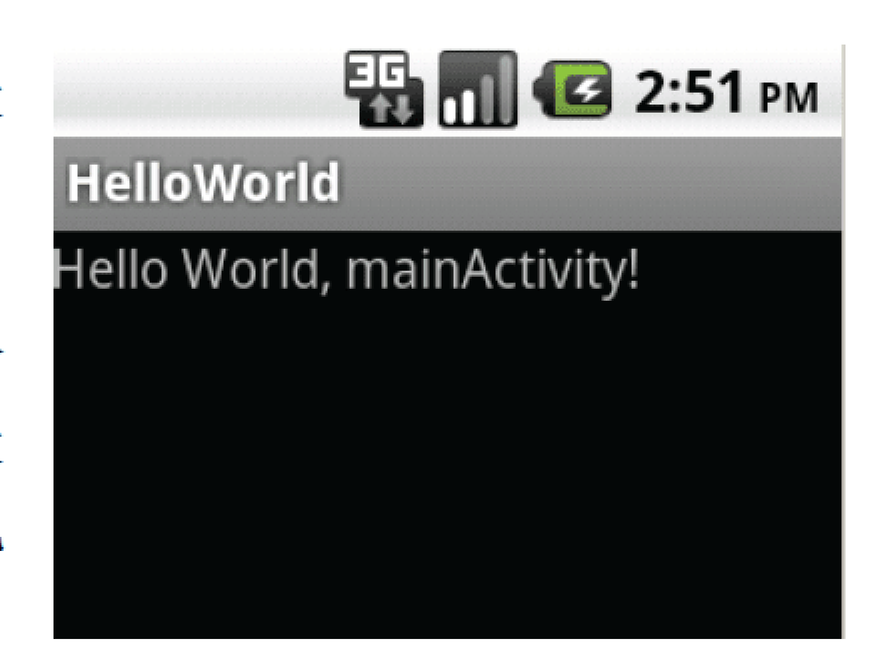


图 5.2　TextView 显示

我们看到的“HelloWorld，mainActivity!”就是一个 TextView，它在屏幕上划出一个矩形区域，并渲染它，最终显示了“HelloWorld，mainActivity!”字符串。要实现 TextView 大体上需要两个步骤。

首先打开 layout 文件夹下的 main.xml 文件，在其中添加一段 xml 配置代码，语法如下：

1. xml代码

```
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/hello"
   />
```

其中：

- ❑ android: layout_width 属性指定了 TextView 渲染的矩形区域的宽。
- ❑ android: layout_height 属性指定了 TextView 渲染的矩形区域的高。
- ❑ android: text 属性指定了 TextView 中显示的文字。这里引用了 String.xml 中的资源，读者也可以将其直接设定为需要显示的字符串，如：

```
android:text="HelloWorld，mainActivity"
```

非常简单的 3 个属性设置之后，我们就完成 TextView 的配置了。配置完成后整体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/hello"
   />
</LinearLayout>
```

完成配置之后如果不与 Java 代码关联，那么配置文件就“英雄无用武之地”了，所以接下来我们需要做的就是将主程序与配置文件“绑定”。实现方法为：Activity.setContent View(int layoutResID)；这里的参数是 layoutResID，读者会说，我还不知道 layout 的资源

ID 呢！事实上，这个 ID 是系统自动分配给每个 layout 的，读者如果想看其具体的值，可以打开 gen 文件夹下的 R.java 文件查看。所以，这里的参数我们可以使用 R.layout.***。

2．Java代码

看一下该程序的代码非常简单，只有几行：

```
package com.wes.helloworld;

import android.app.Activity;
import android.os.Bundle;

public class mainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

这里可以看到，程序的主体只进行了一项工作，就是将程序和名称为 main 的 xml 文件关联起来。其中的 setContentView(R.layout.main);就完成了关联功能。

也许学到这里，很多读者会想，TextView 的确很简单呢。但是在这里要告诉大家的是，TextView 作为一个最基础的 Widget 怎么可能如此简单呢？相信作为开发人员的你同样也无法忍受一个 Widget 只有些许功能吧。

5.2.2　TextView 中的一些功能

接下来要为大家介绍的是一些 TextView 中常用的属性。

1．android:textSize

设置字体大小，如设置为 20dp，其语法形式为：

```
android:textSize="20dp"
```

效果如图 5.3 所示。

2．android:background

设置 TextView 的背景颜色，如设置为白色，其语法形式为：

```
android:background="#FFFFFFFF"
```

3．android:textColor

设置 TextView 的字体颜色，如设置为黑色，其语法为：

```
android: textColor="#000000"
```

将第 2、3 个属性合并就可以设置为经典的“白纸黑字”了，效果如图 5.4 所示。

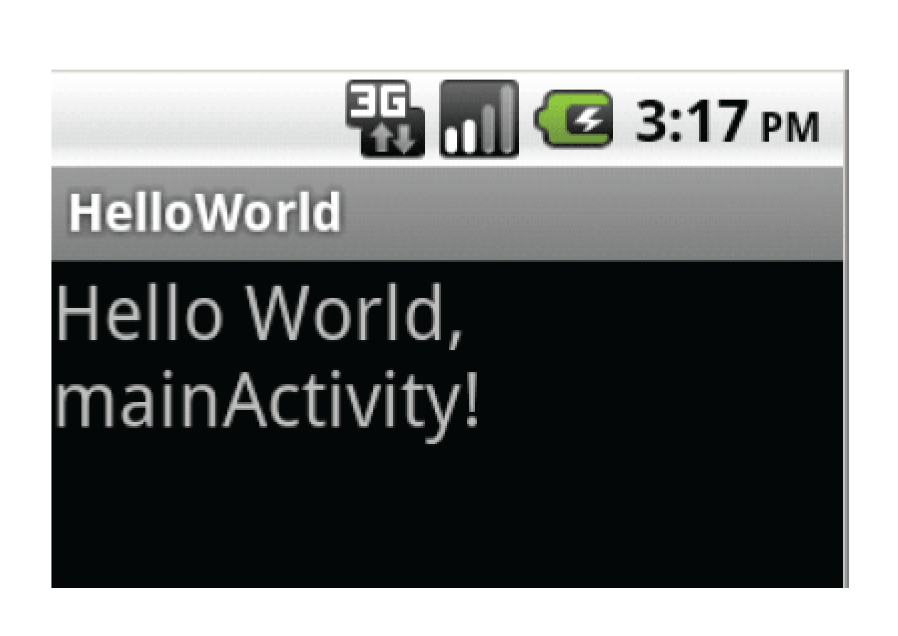


图 5.3　设置字体大小

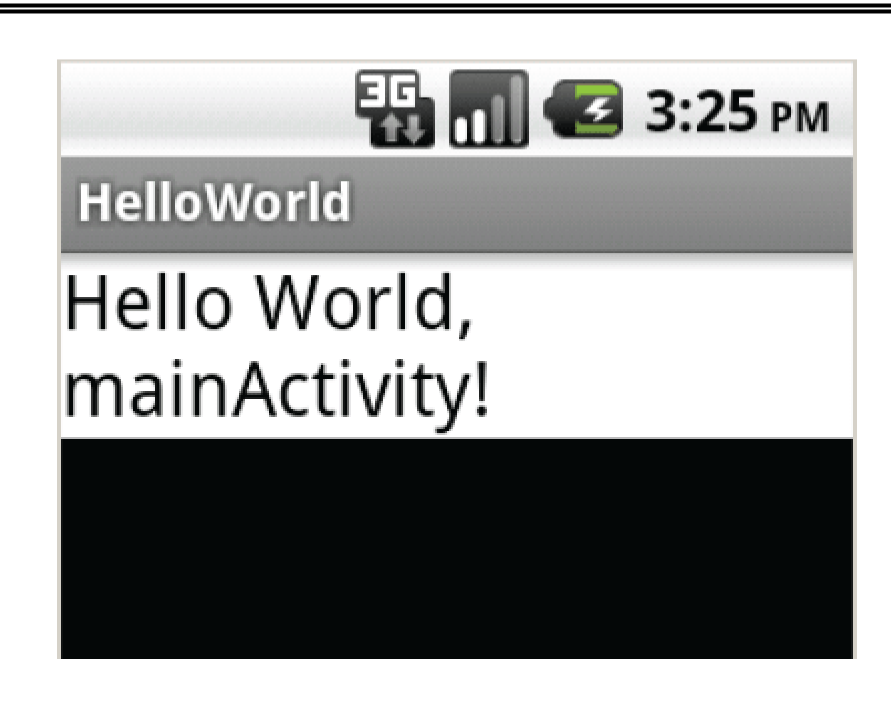


图 5.4　“白纸黑字”的 TextView

4. android:ems

设置 TextView 宽度为 N 个字符的宽度。例如，设置为 10 个 em 宽，语法为：

```
android:ems="10"
```

那为什么这里的属性叫做 ems 呢？因为 Textview 的宽度由单位 em 来决定而不是像素。em 在印刷业中被广泛使用，一个 em 表示一种特殊字体的大写字母 M 的高度。这个属性可以帮助我们很好的设置 TextView 的宽度，而不必关心字体的具体尺寸大小。

5. android:lines

设置文本的行数，如设置为 3，其语法为：

```
android:lines="3"
```

这样 TextView 就将显示 3 行，即使第三行没有字，效果如图 5.5 所示。

假如，你的文字有 4 行，而只显示了 3 行，会出现什么情况呢？我们可以做一个实验，在 android:text 属性中设置 4 行数据，在 android:lines 属性中设置 3 行数据，得到的效果如图 5.6 所示。

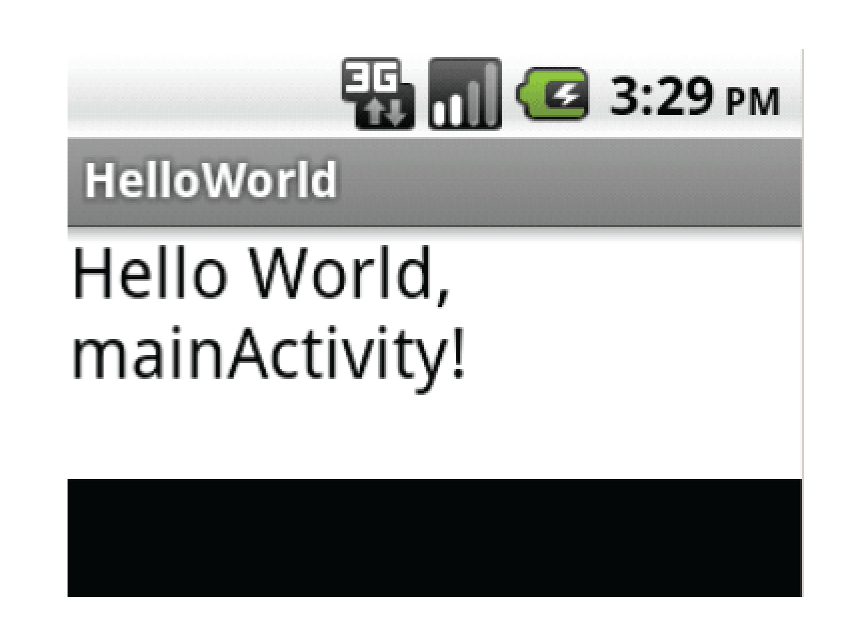


图 5.5　设置显示 3 行

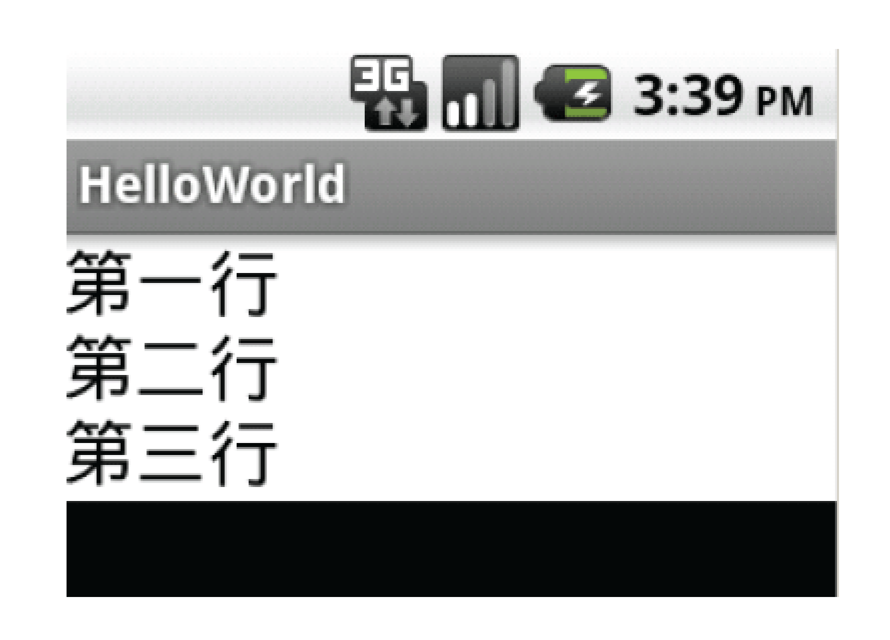


图 5.6　超出范围被切割

从图 5.6 中，我们得出结论：在 TextView 的显示中，超出 TextView 设置大小的部分会被舍弃。那很多读者会产生疑问：这样的话岂不是使用时有很多限制？不要担心，作为一个如此重要的类，必定有解决的方法：滚动！没错，既然在属性中设定了高和宽，那么许多情况下，这一部分的区域是不够显示的，这个时候 TextView 提供的解决办法就是滚动了。

“滚动”我们稍后会有详细的讲解，我们先提出一些较为方便的解决办法，如通过 android:ellipsize 属性解决。

6．android:ellipsize

为了测试该属性，我们将内容设置为：“这里的字符串很长很长很长非常长”。这样在 TextView 区域中将无法完全显示。这时我们将 android:ellipsize 属性值设置为 start，其语法为：

```
android:ellipsize="start"
```

效果如图 5.7 所示。

此处在第二行的开始处将“很长”这两个字省略了。或者我们可以将属性设置为 end，效果如图 5.8 所示。

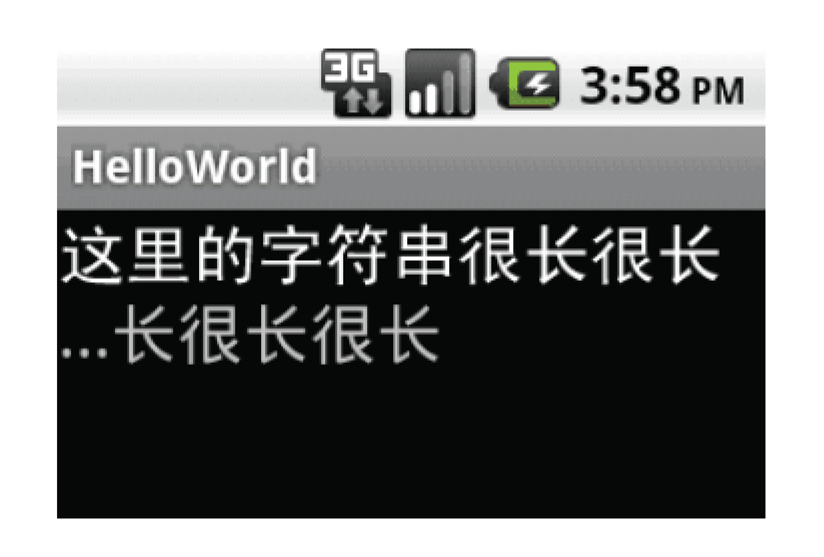


图 5.7　android:ellipsize=“start”

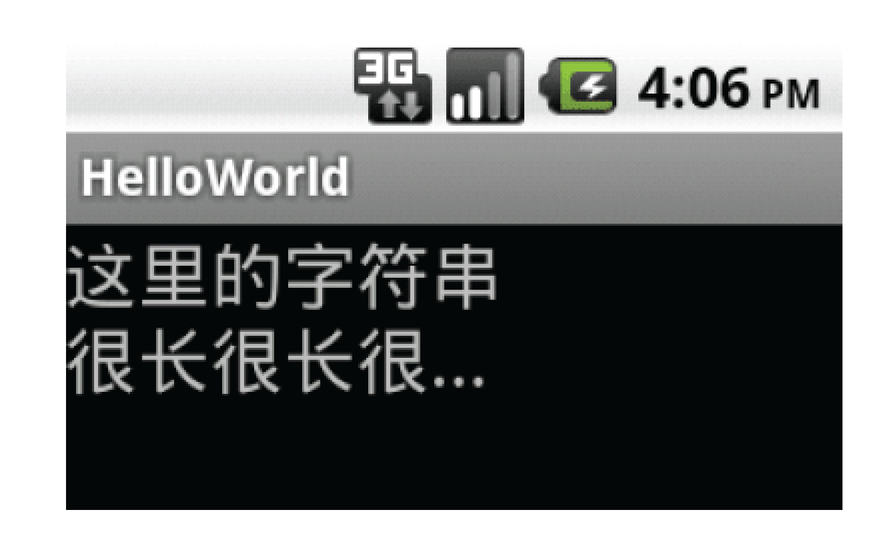


图 5.8　android:ellipsize=“end”

这里在第二行的末尾处将“长非常长”等字符串省略了，我们还可以设置为 middle，顾名思义就是在中间处省略，效果如图 5.9 所示。

最后还有一种选择，就是设置为 marquee 了，也就是平常所说的跑马灯，效果如图 5.10 所示。



图 5.9　android:ellipsize=“middle”



图 5.10　android:ellipsize=“marquee”

注意，这里要将

```
android:focusable="true"
android:focusableInTouchMode="true"
```

两个属性设为 true，跑马灯需要获得焦点才可以进行滚动。

TextView 的一些重要属性就先介绍到这里。

5.2.3　使用可滚动的视图——ScrollView

接下来要介绍的是另一个重要的 Widget——ScrollView。将 TextView 与 ScrollView 结合起来就可以实现页面的滚动了。

ScrollView 也是一个 ViewGroup 的实现类，当需要在一定的区域内显示更多的内容时，我们可以将 View 添加入 ScrollView 中。接下来就来看一下滚动界面的实现。

1. xml配置文件

首先我们看一下 xml 文件的配置信息：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<ScrollView
     android:id = "@+id/sv"
     android:layout_width="400px"
    android:layout_height="150px"
     >
<TextView
     android:id="@+id/textview1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_conten"
     android:textSize="20dp"
    />
</ScrollView>
</LinearLayout>
```

我们看到使用 ScrollView 时只需在 TextView 外“包裹”一层 ScrollView 就可以了。但是要注意的是，当需要改变 View 渲染的矩形区域的大小时，需要设置 ScrollView 的 android:layout_width 和 android:layout_heigh 属性。加入设置 TextView 的这两种属性无法产生理想效果，因为它会被外层包裹的 ScrollView 的这两种属性覆盖。

2. Java代码

Java 部分的代码同样非常简单，首先通过 findViewById()方法找到 xml 文件中配置好的 vTextView 对象，接着可以设置一些属性，如设置背景颜色、字体颜色等：

```
package com.wes.helloworld;

import android.app.Activity;
import android.graphics.Color;                          //导入颜色类
import android.os.Bundle;
import android.widget.TextView;                         //导入 TextView 类

public class mainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView tv = (TextView) findViewById(R.id.textview1);
                                                        //得到 TextView 对象
        tv.setBackgroundColor(Color.WHITE);             //设置背景色
        tv.setTextColor(Color.BLACK);                   //设置字体颜色
```

```
        tv.setText("第一行"+"\n"+"第二行"+"\n"+"第三行"+"\n"+        //设置内容
                  "第四行"+"\n"+"第五行"+"\n"+"第六行"+"\n");
    }
}
```

从本例中可以看出，通过 Java 代码同样可以设置 View 的各个属性，这样的好处是灵活方便，坏处是不便于管理，没有将视图层和逻辑处理层分开。这里设置颜色使用了 TextView.setTextColor(int color)方法，而 int 型的参数我们使用了 graphics.Color 类中的静态常量，这里使用了黑和白，当然还可以设置为其他几种常见的颜色，如蓝、绿、黄等。

实现了两部分的代码后，可以得到的效果图如图 5.11 所示。

到这里，文字的显示就先告一段落，当然这里因为篇幅有限，只是讲解了极少部分知识，大家若要融会贯通还需多挖掘、多实践。

图 5.11　ScrollView 的使用

5.2.4　文字的编辑

学习完文字的显示后，我们再来学习一下文字的编辑。一个应用程序必然需要人机进行交互，如最简单的一个登录界面。我们需要在编辑框中输入账号、密码，然后单击“确定”按钮，完成登录操作。接下来将要学习的就是 EditText，也就是常用的编辑框。

EditText 是 TextView 的子类，所以基本上 TextView 的属性，同样可以作用于 EditText 上。可以将 EditText 理解为可编辑的 TextView。使用 EditText 时，需要编辑如下的 xml 文件：

1．xml配置文件

我们来观察一下 EditText 的配置文件有何玄机：

```
<EditText
    android:id="@+id/edit"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
    android:textSize="20dp"
 >
 </EditText>
```

我们看到只要将<TextView>标签换成<EditText>标签就可以了，有了使用 TextView 的基础后再来学习使用 EditText 就非常方便了。

2．Java代码

Java 部分的代码基本与 TextView 相同：

```
package com.wes.helloworld;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.widget.EditText;                                //导入 EditText 类
```

```
public class mainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        EditText et = (EditText) findViewById(R.id.edit);
                                                        //得到 EditText 对象
        et.setBackgroundColor(Color.WHITE);             //设置背景色
        et.setTextColor(Color.BLACK);                   //设置字体颜色
        et.setText("第一行"+"\n"+"第二行"+"\n"+"第三行"+"\n"+  //设置默认内容
                "第四行"+"\n"+"第五行"+"\n"+"第六行"+"\n");
    }
}
```

这部分代码应该没有什么难度，那么接下来我们可以看一下效果图，如图 5.12 所示。



图 5.12　EditText 效果图

当 EditText 获得焦点时，会自动弹出软键盘，以供用户输入。当不需要键盘自动弹出时，我们可以将 EditText 的焦点获得设置为 false，这样就切断了软键盘的弹出了。顺带一提的是：EditText 自带滚动，我们无需在外层包裹一个 ScrollView 了。

好了，讲解到这里文字处理相关的组件就介绍的差不多了，希望大家可以结合本书多动手、实践体会，在实践过程中必然会遇到很多问题，多思考、多查阅必然能找到解决的办法。下一小节，将介绍另一个经常使用组件——按钮。

5.2.5　使用按钮——Button

本小节将介绍的 Widget 组件，在平时的操作中扮演着很特殊的角色——终结者。一般情况下，一次人机的交互都以一个按钮的单击事件结束，所以学习使用按钮也是开发中的必需。与之前我们学习的三类组件不一样的是，按钮组件必须要在 Java 代码中实现一些逻辑上的处理，也就是说我们必须要为按钮添加单击的响应事件，否则，这个按钮也就形同虚设。

1．Button的使用

接下来我们就开始按钮的学习，首先我们需要在 xml 配置文件中添加如下 xml 代码：

```
<Button
     android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>
```

当然这里只是最简单的配置，我们可以在属性中添加各种我们需要的信息，如 android:id 属性、android:text 属性等等。

2．Java部分代码

在 Java 部分代码中我们首先得到按钮对象，然后添加其单击事件的响应：

```
package com.wes.button;

import ......                                          //省略部分导入包
import android.view.View.OnClickListener;             //导入监听器类
import android.widget.Button;                         //导入按钮类

public class mainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Button   btn= (Button)  findViewById(R.id.btn); //获得按钮操作对象
        btn.setOnClickListener(new OnClickListener()
         {

              @Override
              public void onClick(View arg0)
              {                                    //这里实现单击后的逻辑处理
              }
         });
    }
}
```

使用 View.setOnClickListener(OnClickListener l)方法来设置监听器，参数是 OnclickListener 接口。在接口中我们需要重写 onClick(View arg0)方法，也就是在该方法内完成需要进行的逻辑处理。

5.2.6　实例——计算器

接下来就结合前一小节中学习的知识写一个简单的计算器。

首先分析一下我们要做的计算器程序，我们需要两个编辑框来输入需要计算的数字，需要一个文本框来显示运算法则，需要一个文本框来显示计算结果，最后需要一个按钮确定操作。接下来我们来看一下效果图，如图 5.13 和图 5.14 所示。



图 5.13　button 单击后



图 5.14　button 单击前

接下来我们照例观察 xml 代码。

1．xml代码

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<EditText
    android:id="@+id/et1"
    android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:paddingTop="10px"
   android:text="3"
    />

<TextView
    android:id="@+id/tv1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:textSize="30sp"
   android:text="乘以"
   />

<EditText
    android:id="@+id/et2"
    android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:paddingTop="10px"
   android:text="3"
    />
<Button
    android:id="@+id/btn"
    android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="确定"
/>

<TextView
    android:id="@+id/tv2"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:textSize="30sp"
   android:text="等于"
   />
</LinearLayout>
```

相信通过上一小节的学习，大家应该能够自行写出本例的配置文件了，这里就不再赘述，重点来看一下 Java 部分的代码。

2．Java代码

在 Java 部分我们首先得到需要操作的组件对象，分别是 et1 编辑框（用来输入第一个参数）、et2 编辑框（用来输入第二个参数）、tv2 文本框（用来显示计算结果）、btn 按钮（用来完成本次操作）。代码片段如下：

```
TextView result = (TextView) findViewById(R.id.tv2);
 EditText  et1  = (EditText) findViewById(R.id.et1);
EditText  et2  = (EditText) findViewById(R.id.et2);
 Button  btn  = (Button)  findViewById(R.id.btn);
```

下一步就是为按钮设置单击的监听事件了，代码如下：

```
btn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                int arg1 = Integer.parseInt(et1.getText().toString());
                                                        //得到第一个参数
                int arg2 = Integer.parseInt(et2.getText().toString());
                                                        //得到第二个参数
                int answer = arg1 * arg2;               //计算结果
                result.append(String.valueOf(answer)); //显示结果
            }
        });
```

这里需要注意的是我们得到的编辑框中的内容是 Editable 类型，我们需要使用 toString()方法将其转成 String 格式，接着使用 Integer.parseInt()方法将 String 型转换为 int 型，这样才能对这两个参数进行计算。最后显示时还需将 int 型的结果转换为 String 类型显示。这里使用了 TextView.append(CharSequence text)方法，作用为在 TextView 的原有内容后继续追加内容。

这里需要注意的是在重写 onClick()方法时，要操作方法外的 Widget 对象需要用 final 关键字修饰该对象。

5.2.7　使用图片按钮——ImageButton

学习到这里，Button 的使用就告一段落了，当然为了使界面更加美观、更加华丽，我们还可以使用另一组件——ImageButton（图片按钮）。当我们希望按钮以图片的形式出现时，可以使用 ImageButton 组件，这样我们的界面就显得更生动了。

1. ImageButton的xml配置文件

ImageButton 的 xml 配置文件与 Button 大体相仿，不同的是需要与事先准备好的图片进行绑定，这里使用了 android:background 属性。

```
<ImageButton
 android:id="@+id/btn"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:background="@drawable/ok">
</ImageButton>
```

这样 xml 文件就编写完毕了。Java 部分的代码与 Button 相同，这里就不再贴出代码了，依旧使用上一个例子，最后的显示如组图 5.15 所示。





图 5.15　imagebutton

这里我们只是将其 background 属性与图片绑定，其实我们还可以使用 xml 文件与 ImageButton 绑定。

首先我们可以在 drawable 文件夹中新建 advancedbutton.xml 文件，在其中设置 seletor 和 item 标签。当然，要配置该 xml 文件我们必须先准备好 4 张图片，如图 5.16 所示。

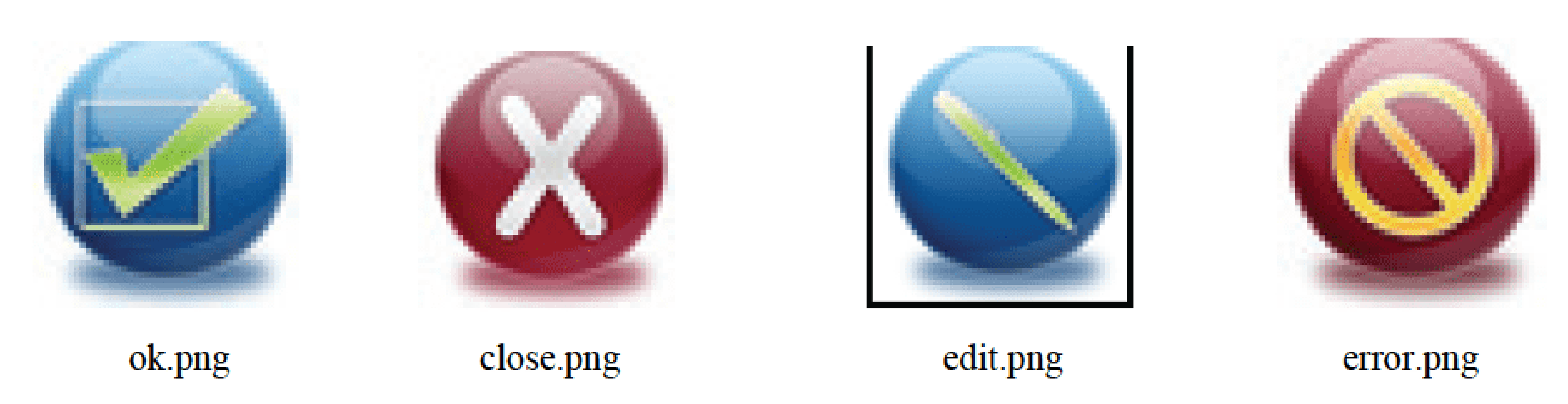


图 5.16　需要使用的图片

2. xml代码

读者请注意观察 selector 标签和 item 标签中的属性配置：

```
<?xml version="1.0" encoding="utf-8"?>

<selector xmlns:android="http://schemas.android.com/apk/res/android">

<item
    android:state_focused="true"              //该属性判断焦点获得与否，下同
    android:state_pressed="false"             //该属性判断是否按下，下同
    android:drawable="@drawable/edit"         //绑定该状态下显示的图片，下同
/>

<item
    android:state_focused="true"
    android:state_pressed="true"
    android:drawable="@drawable/close"
/>

<item
```

```
    android:state_focused="false"
    android:state_pressed="true"
    android:drawable="@drawable/error"
/>

<item
    android:drawable="@drawable/ok"                //默认为两属性皆为 false
/>
</selector>
```

可以看出一个按钮一共有 4 种状态，在本 xml 中都进行了配置。

3. Java代码

Java 代码部分与上例相同，不需做任何修改，这也是使用 xml 文件配置界面的好处了：当我们希望界面改变时只需改动 xml 部分而不需改变 Java 部分的代码。

最后效果如图 5.17 所示。



默认情况



获得焦点但未按下



获得焦点并按下



按下但失去焦点

图 5.17　ImageButton

ImageButton 就讲解到这里，希望读者可以灵活应用，以写出更绚丽的界面。下一小节我们将讲解复选框（CheckBox）、单选框（RadioGroup）以及下拉列表框（Spinner）的使用。

5.2.8　使用复选框——CheckBox

上面章节中已经完成了一些简单的人机交互操作，理论上从 EditText 就可以获得一切我们希望用户提供的信息。但是，无疑这么做是非常不友好的，甚至是繁琐而又令人厌恶的。这时，可以提供一些选项以供用户选择，如年龄、性别等。

1．CheckBox的使用

CheckBox 常被用于这样的情景下：下载软件时弹出一个许可协议，选择是否同意；给出一个列表，勾选想要的选项；总之，针对某个选项你希望用户给出“是”或“否”的操作时，就可以用它了。

要使用一个 CheckBox 组件首先学习其 xml 文件的配置方法。

2．xml代码

使用 CheckBox 时需使用<CheckBox>标签：

```
   <CheckBox
   android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="同意"
/>
```

其中 android:text 属性显示了该复选框的提示信息，读者可以任意修改为自己想要的提示信息，如“全选”、“全部同意”等。

3．Java代码

Java 代码相对也非常简单，同样需要实现 View.setOnClickListener(OnClickListener l)方法，以监听是否被单击，在实现 OnClick(View arg0)方法时，可使用 CompoundButton.isChecked()方法来判断该选项是否被选中。

5.2.9　实例——请同意本协议

1．小示例——同意协议

接下来看一个小例子：假设有一个协议需要你选择是否同意，选中“同意”则显示“您已同意该协议”，未选中则显示“您未同意该协议”。需求非常简单，使用 CheckBox 正是相得益彰。

首先我们编写 xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
```

```
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
     <TextView
     android:id="@+id/tv1"
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:text="你是否同意该条款，同意请勾选"
    android:textSize="30sp"
    />

    <CheckBox
    android:id="@+id/cb"
    android:layout_width="wrap_content"
    android:layout height="wrap content"
    android:layout marginTop="150px"
    android:layout marginLeft="110px"
    android:text="同意"
    />

    <TextView
     android:id="@+id/tv2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="30sp"
    />

</LinearLayout>
```

这里我们在 CheckBox 的属性中使用了 android:layout_marginTop 和 android:layout_marginLeft，其目的是给 CheckBox 指定坐标，分别为距离屏幕顶部的长度和距离屏幕左边的长度。

接下来分析 Java 代码：

```
package com.wes.checkbox;

import ......
import android.view.View.OnClickListener;          //导入 OnClickListene 接口
import android.widget.CheckBox;                    //导入 CheckBox 类

public class Checkbox extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final TextView tv = (TextView) findViewById(R.id.tv2);
                                                   //得到 TextView 的对象
        final CheckBox cb = (CheckBox) findViewById(R.id.cb);
                                                   //得到 CheckBox 的对象

        cb.setOnClickListener(new OnClickListener()
         {
            @Override
            public void onClick(View arg0)
            {
                // TODO Auto-generated method stub
                if (cb.isChecked())                //判断是否被选中
```

```
                    {
                        tv.setText("您同意了该条款");
                    }
                    else
                        tv.setText("您未同意该条款");
                }
            });
        }
}
```

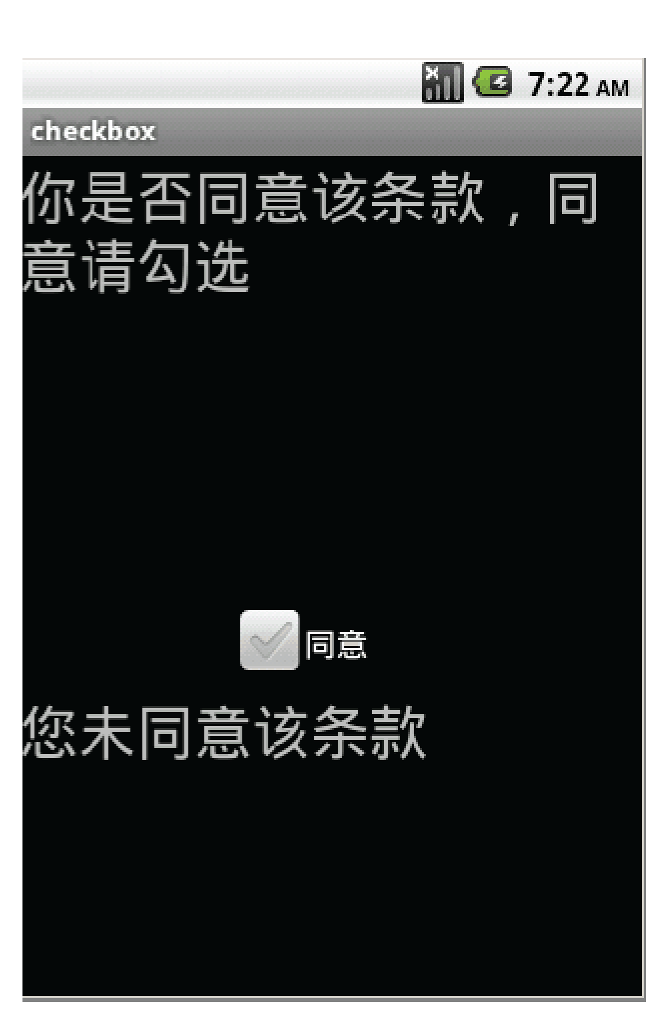
此处代码非常简洁明了，相信读者可以完全理解了。接下来就看看运行效果吧，如图 5.18 所示。



初始界面



面选中后



取消选中

图 5.18　CheckBox 的使用

2．丰富示例——请选择兴趣爱好

当然上面只是一个最简单的应用，我们还可以利用它做什么呢？如选择你的兴趣爱好！我们可以再看一个例子，使用多个 CheckBox 以达到多选的效果。

这个例子的假设场景是这样的：当你注册一个账户时，需要填写你的兴趣爱好以丰富你的个人资料，这个时候我们也可以使用 CheckBox 达到目的。

首先是 xml 代码，这里创建了 3 个 CheckBox 以供选择：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent"
   >
    <TextView
    android:id="@+id/tv"
   android:layout width="fill parent"
   android:layout height="wrap content"
   android:text="请选中您的兴趣爱好"
   android:textSize="30sp"
   />
   <CheckBox
```

```
    android:id="@+id/cb1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="5px"
    android:layout_marginLeft="20px"
    android:text="阅读"
    />
    <CheckBox
    android:id="@+id/cb2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="5px"
    android:layout_marginLeft="20px"
    android:text="游泳"
    />
    <CheckBox
    android:id="@+id/cb3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="5px"
    android:layout_marginLeft="20px"
    android:text="玩网游"
    />
    ......                                          //这里省略了部分组件的xml代码
</LinearLayout>
```

这里省略的是 TextView 的编写，读者如需了解请自行查看源码。

接下来观察 Java 代码，这里我们使用了另一个监听器：CompoundButton.OnCheckedChangeListener，希望读者注意学习。

```
package com.wes.checkbox2;

import ......
import android.widget.CheckBox;
import android.widget.CompoundButton;           //导入复合按钮类
import android.widget.CompoundButton.OnCheckedChangeListener;
                                                //导入复合按钮类的监听接口

public class Checkbox_2 extends Activity {
     private TextView tv1;
     private TextView tv2;
     private TextView tv3;
     private CheckBox cb1;
     private CheckBox cb2;
     private CheckBox cb3;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

       tv1 = (TextView) findViewById(R.id.tv1); //此处获得需要使用的View的对象
       tv2 = (TextView) findViewById(R.id.tv2);
       tv3 = (TextView) findViewById(R.id.tv3);
       cb1 = (CheckBox) findViewById(R.id.cb1);
       cb2 = (CheckBox) findViewById(R.id.cb2);
       cb3 = (CheckBox) findViewById(R.id.cb3);
     }
```

```
    private OnCheckedChangeListener listener = new OnCheckedChangeListener()
    {
        @Override
     public void onCheckedChanged(CompoundButton btn, boolean isChecked)
     {
        //此处实现接口的功能实现
     }
    };
}
```

到这里整体设计完成，接下来就实现其具体的功能了：

```
if (cb1.isChecked())
        {
            tv1.setText("阅读");
        }else
            tv1.setText("");
        if (cb2.isChecked())
        {
            tv2.setText("游泳");
        }else
            tv2.setText("");
        if (cb3.isChecked())
        {
            tv3.setText("玩网游");
        }else
            tv3.setText("");
```

这里的判断语句想必对读者应该不会造成困扰，不过需要说明的是笔者为了简化判断，所以创建了多个 TextView 用于显示，读者如果有兴趣可以只使用一个 TextView 同样可以完成这个例子，当然判断的时候会比较繁琐一些。

最后我们需要为这 3 个 CheckBox 设置一下监听器，否则它们是无法工作的：

```
cb1.setOnCheckedChangeListener(listener);
cb2.setOnCheckedChangeListener(listener);
cb3.setOnCheckedChangeListener(listener);
```

代码编写完毕我们来看一下效果图，如图 5.19 所示。

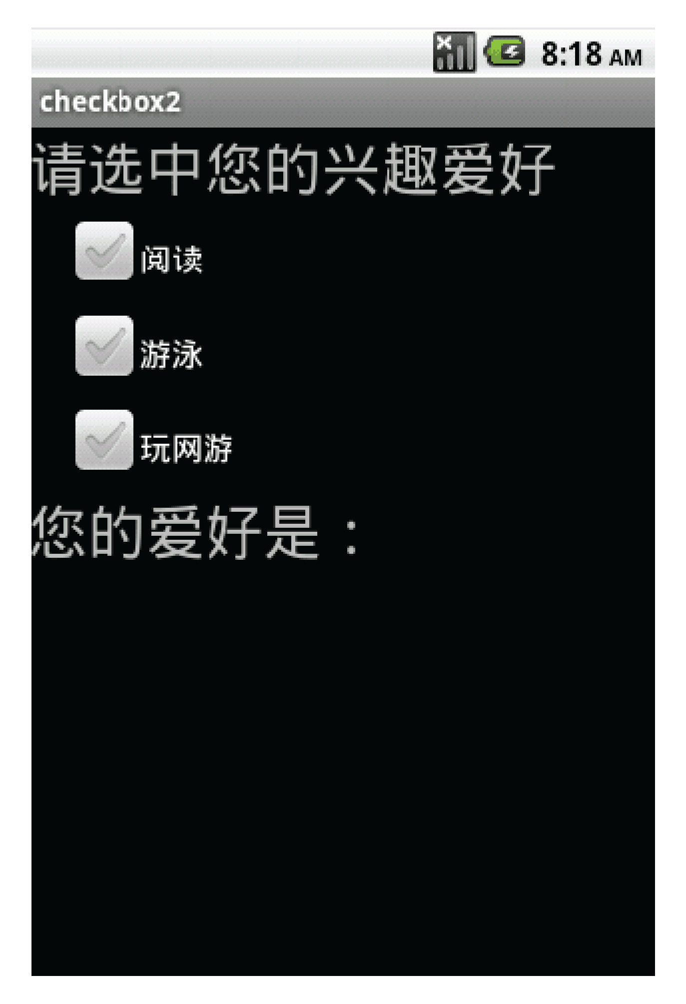








图 5.19　CheckBox 的多选使用

到这里，关于 CheckBox 的学习就结束了，接下来我们将学习另一个功能类似的 Widget——RadioGroup，也就是常说的单选框。

5.2.10　使用单选框——RadioGroup

之前我们学习了使用 CheckBox 进行多选，那么当我们希望用户只能选择其一的时候该怎么办呢？这个时候，我们可以使用一个新的 Widget——单选框，也就是 RadioGroup。

照例我们先学习其 xml 代码：

```
<RadioGroup
android:layout width="fill parent"
android:layout height="wrap content"
>
 <RadioButton
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 />
 <RadioButton
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
/>
</RadioGroup>
```

由 xml 代码我们可以看出，RadioGroup 是个 View 容器，其中需要添加选项。这里我们添加了两项，当然读者可以添加更多，但是最后只能选择其中一个选项。

在 Java 代码中我们需要使用 RadioGroup.setOnCheckedChangeListener(OnCheckedChangeListener listener)方法设置 RadioGroup 的监听器。在实现 OnCheckedChangeListener 接口时，可以判断该 RadioGroup 内包含的 RadioButton 的选中状态。

5.2.11　实例——请选择性别

接下来看一个例子，通过例子更好地学习 RadioGroup 的使用。

首先是场景假设：同样是用户注册时，需要用户填写性别，如果使用之前的组件都不能很好地达到预期的效果，这个时候我们刚学习 RadioGroup 就可以大显身手了。首先观察

xml 代码。

1. xml代码

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
    <TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="请选择您的性别："
   android:textSize="30sp"
   />

   <RadioGroup
   android:id="@+id/radioGroup"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   >
    <RadioButton
    android:id="@+id/radio1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="绅士"
    android:textSize="20sp"
    />
    <RadioButton
    android:id="@+id/radio2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="淑女"
    android:textSize="20sp"
    />
    <RadioButton
    android:id="@+id/radio3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="保密"
    android:textSize="20sp"
    />
   </RadioGroup>
......                                                      //省略部分组件
</LinearLayout>
```

这里我们加入了 3 个 RadioButton，分别是“绅士”、“淑女”还有“保密”。另外还新建了两个 TextView 用来显示信息，这里省略，读者可查看源代码。

2. Java代码

这里需要注意的是，我们是对 RadioGroup 进行监听而非对其中的一个按钮进行监听：

```
package com.wes.radiogroup;

import ......
import android.widget.RadioButton;                          //导入 Radio 按钮类
import android.widget.RadioGroup;                           //导入 Radio 容器类
```

```
import android.widget.RadioGroup.OnCheckedChangeListener   //导入监听器

public class Radiogroup extends Activity {
    /** Called when the activity is first created. */
     RadioGroup  rg ;
     RadioButton rb1;
     RadioButton rb2;
     TextView    tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        rg = (RadioGroup) findViewById(R.id.radioGroup);
                                            //获得需要操作的 View 的对象
        rb1= (RadioButton)findViewById(R.id.radio1);
        rb2= (RadioButton)findViewById(R.id.radio2);
        tv = (TextView)    findViewById(R.id.tv);

        rg.setOnCheckedChangeListener(new OnCheckedChangeListener()
         {
             @Override                      //实现 OnCheckedChangeListener 接口
             public void onCheckedChanged(RadioGroup arg0, int arg1)
             {
                 // 这里完成逻辑处理
             }
         });
    }
}
```

完成了整体的设计，接下来就是一个简单的功能实现了，代码如下：

```
             public void onCheckedChanged(RadioGroup arg0, int arg1)
             {
                 // TODO Auto-generated method stub
                 if (rb1.isChecked())
                     tv.setText("绅士");
                 else if (rb2.isChecked())
                     tv.setText("淑女");
                 else
                     tv.setText("保密");
             }
```

写到这里又一个例子完成了，赶快看一下实际运行的效果吧，如图 5.20 所示。



图 5.20　RadioGroup 的使用

通过这个例子相信读者应该能掌握 RadioGroup 的使用了，下一小节我们将要学习的 Widget 是下拉列表——Spinner。

5.2.12　使用下拉列表框——Spinner

RadioGroup 固然好用，但是毕竟手机的界面是有限的，就像上一小节的例子在实际开发中无疑是奢侈的。这时，Spinner 组件就华丽地登场了。Spinner 组件只需很小的一块区域要能显示，当你单击时，会弹出一个列表选项。相信大家在使用电脑时肯定经常使用下拉列表，笔者就不再做过多的描述了。

首先我们观察 xml 文件的配置：

```
    <Spinner
    android:layout_width=" wrap_content "
    android:layout_height="wrap_content"
/>
```

同样的道理，使用 Spinner 只需使用<Spinner>标签就可以了。当然在实际的使用过程中还需进行一些其他的配置，我们可以在接下来的实例中具体讨论。

在 Java 代码中，使用 Spinner 需要进行 4 个步骤：

（1）获取 Spinner 对象。

（2）创建 Adapter。

（3）为 Spinner 对象设置 Adapter。

（4）为 Spinner 对象设置监听器。

其中创建 Adapter 又分为两步（至少）：

（1）新建 Adapter 对象。

（2）设置下拉视图的资源。

所以实际上，一共需要 5 个步骤或者更多，接下来我们就仔细分析每个步骤的做法和功能。

1．获取Spinner对象

通过 Activity.findViewById(int id)方法获取 View 的对象现在对于读者肯定没有什么问题了。这一步就浅尝辄止了。

2．创建Adapter

最简单的，我们可以通过 ArrayAdapter.ArrayAdapter(Context context, int textViewResourceId, List<String> objects)构造方法创建新的 Adapter 对象。

这里需要 3 个参数：

（1）context 上下文关系，宽泛地说就是这个 Adapter 属于哪个 Activity，属于哪个应用程序。

（2）textViewResourceId，顾名思义，这个参数是 TextView 的资源 Id，我们可以自己写一个 TextView，当然也可以使用系统自带的 TextView，这在之后的实例中都有讲解。

（3）最后一个参数是你需要向下拉列表中添加的数据，可以是一个静态的 Strin 数组，

也可以是一个动态的 List<String>。这在后面的实例中同样会有说明。

3．为Adapter设置下拉视图的资源

使用 ArrayAdapter.setDropDownViewResource(int resource)方法设置下拉视图资源，同样的这里我们可以自己配置或者使用系统提供的资源。

4．为Spinner对象设置Adapter

通过 AbsSpinner.setAdapter(SpinnerAdapter adapter)方法可以很方便地将 Spinner 与 SpinnerAdapter 关联起来。

5．为Spinner对象设置监听器

通过 AdapterView.setOnItemSelectedListener(OnItemSelectedListener listener)方法设置监听器。注意这里的参数是一个 OnItemSelectedListener 接口，实现这个接口需要重写两个方法，分别是：

（1）void onItemSelected(AdapterView<?>parent,View view, int position，long id)，这个方法中可以完成当选项被选中时要做的处理。我们看到这个方法有 4 个参数，其意义分别为：

- AdapterView<?>parent，这个参数的意义类似于 context，只是范围较小，是指你当前操作的 AdapterView，从对象名 parent 也可看出一些端倪，即父视图。
- View view，这个参数是你具体单击的那个 TextView 对象。
- int position，这个参数的意义是你单击的那个 view 在整个 AdapterView 中的位置，如第一个则 position 为 0。
- long id，这个参数在实际的编程中使用较少，其意义为被单击的 view 的 id。

（2）void onNothingSelected(AdapterView<?>parent)，这个回调函数在该 AdapterView 中没有选项时被调用。这时只有一个参数 AdapterView，因为其内的选项为空了，当然没有其他参数了。

好了，到这里一个 Spinner 对象就可以被正常使用了，当然你也可以对其进行更多的设置，笔者这里因为篇幅有限就不再过多讲解了。

5.2.13　实例——请选择工作年限

通过以上理论知识的学习，读者也许还不是很明白，没有关系，通过实例可以帮你很快地掌握这个组件的使用。这个实例的功能为：提供给用户一个下拉列表，其中用户可以选择自己已经工作的年限，通过单击其中的选项完成选择，并且最后在页面上要有所展示。

1．xml代码

首先编写 main.xml 文件的代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
     <TextView
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:text="请选择工作年限："
    android:textSize="20sp"
    />
    <Spinner
    android:paddingTop="10px"
    android:id="@+id/spin"
    android:layout_width="fill_parent"
    android:layout_height="50sp"
    />
......                                                      //此处省略部分组件
</LinearLayout>
```

接下来还需要编写下拉列表的视图资源，即每个 Item 的 TextView，我们将它命名为 dropdown.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<TextView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tv1"
    android:layout_width="fill_parent"
    android:layout_height="20sp"
    android:singleLine="true"
    style="?android:attr/spinnerDropDownItemStyle"
/>
```

这里笔者使用系统自带的样式，也就是 style="?android:attr/spinnerDropDownItem Style"，读者也可以自行编写样式以求画出更好看的下拉列表。

2. Java代码

```
package com.wes.spinner;

import ......                                                //此处省略部分导入包
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;

public class spinner extends Activity {
     private Spinner spinner;                               //声明需使用的对象
     private TextView tv;
     private ArrayAdapter<String> adapter;
     private static final String[] years = {"小于 1 年","1 年~3 年","3 年~5 年
","5 年以上"};    //Spinner 的数据源
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        spinner = (Spinner) findViewById(R.id.spin);        //获得 Spinner 对象
        tv = (TextView) findViewById(R.id.tv);
        adapter = new ArrayAdapter<String>(this,            //新建 Adapter 对象
             android.R.layout.simple_spinner_item, years);
        adapter.setDropDownViewResource(R.layout.dropdown);
```

```
                                                    //设置下拉视图资源
       spinner.setAdapter(adapter);                 //设置 Adapter

       spinner.setOnItemSelectedListener(new OnItemSelectedListener()
        //设置监听器
        {
            //这里实现接口
        });
    }
}
```

依照之前讲解的 5 个步骤，是不是思路清晰了很多呢？其中新建 Adapter 时，资源使用系统自带的 android.R.layout.simple_spinner_item，读者也可以自己定义一个。

接下来就完成接口定义的方法的重写：

```
@Override
            public void onItemSelected(AdapterView<?> parent, View view,
            int position, long id)
            {
                String seleted = years[position];
                tv.setText(seleted);
                parent.setVisibility(View.VISIBLE);
            }

            @Override
            public void onNothingSelected(AdapterView<?> arg0)
            {
                tv.setText("您没有选择");
            }
```

这样一个实例就编写完毕了，运行后的效果如图 5.21 所示。



图 5.21　Spinner 的静态使用

也许到这里很多读者就满足了，认为已经学会使用 Spinner 组件了，但是，如果我们需要动态添加和删除下拉列表框中的内容呢？答案就是在新建 Adapter 时将最后的数据源参数改为 List<String>类型，我们只需将上一个例子稍作修改就可以了。

5.2.14　实例——动态修改 Spinner 项

首先是 xml 代码。

1．xml代码

在 xml 代码中添加一个 Button：

```
<Button
android:id="@+id/btn"
android:paddingTop="10px"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
 android:text="添加"
/>
```

单击这个 Button 就在下拉菜单中添加一个选项。

2．Java代码

Java 代码中，我们为 Adapter 配置资源时都使用系统自带资源，看看是什么样的状况：

```
package com.wes.spinner;

import java.util.ArrayList;

import ......
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;

public class spinner extends Activity {
    /** Called when the activity is first created. */
     private Spinner spinner;
     private TextView tv;
     private Button  btn;
     private ArrayAdapter<String> adapter;
     private static final String[] years = {"小于 1 年","1 年~3 年","3 年~5 年
","5 年以上"};
     private ArrayList<String> array = new ArrayList<String>();
                                   //新建一个 ArrayList 存放数据
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        spinner = (Spinner) findViewById(R.id.spin);
        tv = (TextView) findViewById(R.id.tv);
        btn = (Button) findViewById(R.id.btn);

        for (int i = 0;i < years.length; i++)
        {
         array.add(years[i]);        //将 String 数组中的数据添加到 ArrayList 中
        }
```

```
        adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple spinner item, array);
                                                //把 Array 添加到 Adapter 中
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_
        dropdown_item);
        spinner.setAdapter(adapter);

        spinner.setOnItemSelectedListener(new OnItemSelectedListener()
         {

            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
            int position, long id)
            {
                String seleted = array.get(position);
                                                //通过 get()方法获得数据
                tv.setText(seleted);
                parent.setVisibility(View.VISIBLE);
            }

            @Override
            public void onNothingSelected(AdapterView<?> arg0)
            {
                tv.setText("您没有选择");
            }
         });

        btn.setOnClickListener(new OnClickListener()
         {

            @Override
            public void onClick(View arg0)
            {
                array.add("10 年以上");           //向 Array 中添加数据
                Toast.makeText(getBaseContext(), "成功添加", Toast.LENGTH_
                LONG).show();
            }
         });
    }
}
```

注意观察粗体部分，我们转换了 Adapter 中的数据源的数据结构，由静态的 String 数组改为了动态的 ArrayList<String>，这样就可以完成选项的动态修改了。单击 Button 按钮后，在选项中添加“10 年以上”选项。

最后运行效果如图 5.22 所示。

这里只是做了一个简单的动态添加，读者可以自行完善，如添加一个 EditText 组件，单击“添加”按钮则获取编辑的内容并添加入下拉列表中。还可以添加一个 Button，单击后从列表中删除一些选项，等等。

关于选择框的组件我们就讲解到这里了，下一小节讲解的内容是进度条 ProgressBar 和拖动条 SeekBar。

5.2.15　使用进度条——ProgressBar

本小节将讲解进度条和拖动条的使用，使用进度条可以很直观地向用户展示程序目前

的运行进程，非常友好；使用拖动条则可以很方便地拖动到用户希望到达的位置。这两个组件都是经常使用的，其目的是使用户操作时感觉更加的直观和方便。













图 5.22　动态修改 Spinner

1．ProgressBar的使用

进度条 ProgressBar 主要用于体现后台运行的进程的完成程度，如下载文件时显示下载进度，程序初始化时显示初始化的完成程度等。在 Android 中有两类进度条：圆形进度条和水平进度条。首先我们看一下 xml 代码。

2．xml代码

```
<ProgressBar
    android:layout_width="wrap_content"
   android:layout_height="wrap_content"
/>
```

使用<ProgressBar>标签就可以创建进度条了，假如不设置其他属性则显示为系统默认的“中号”的圆形进度条。那么水平进度条呢？非常简单，只需在<ProgressBar>标签下添加一个 style 属性就可以了，其属性值设为：

```
style="?android:attr/progressBarStyleHorizontal"
```

这样就可以将进度条样式从圆形进度条改为水平进度条了。

3. Java代码

接下来就是 Java 代码的编写了，Java 部分的代码也非常简单，第一步依旧是获得其操作对象，接下来：假如你不希望体现具体的进度，只是想告诉用户程序正在后台运行，那么你就不需要再进行任何操作了；假如你希望体现出具体的进度，那么就需要使用如下的方法了：

```
ProgressBar.setProgress(int progress)
```

这里的 int 型参数为指示 ProgressBar 显示到哪一个值。例如，我们在新建水平进度条时将进度条的最大值定为 100，那么我们可以把 ProgressBar 看成均分的 100 份，使用 ProgressBar.setProgress(50)时，相当于将进度条拖动到第 50 份。具体的用法我们会在后面的实例中进行讲解。

5.2.16 实例——动态修改进度条

通过之前的理论铺垫，接下来我们实际编写程序以巩固加深。我们将在界面上显示 5 种进度条，分别是标题栏的微型进度条、小型进度条、默认型进度条、大型进度条以及水平进度条。然后通过一个按钮的单击事件改变进度条的进度显示。首先是 xml 代码。

1. xml代码

请注意各个 ProgressBar 的 style 属性。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
    <TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="进度条"
   />
    <ProgressBar                                              //默认为中号
        android:layout_width="wrap_content"
       android:layout_height="wrap_content"
    />

    <ProgressBar                                              //设置为小号

       style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
```

```
        android:layout height="wrap content"
    />

    <ProgressBar                                          //设置为大号
        style="?android:attr/progressBarStyleSmall"
         android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />

    <ProgressBar                                          //设置为水平
        android:id="@+id/bar"
        style="?android:attr/progressBarStyleHorizontal"
         android:layout width="200dip"
        android:layout height="wrap content"
        android:max="100"
/>
    ......
</LinearLayout>
```

读者朋友们有没有发现什么？这里只有 4 个，为什么之前介绍是 5 个呢？因为还有一个显示在标题栏中的进度条只需在 Java 部分用两句代码就可以完成了。

2．Java代码

Java 部分的代码相对也比较简单，首先是整体设计：

```
package com.wes.progbar;

import ......                                  //省略部分导入包
import android.view.Window;
import android.widget.ProgressBar;
import android.widget.Toast;

public class ProgBar extends Activity {
   /** Called when the activity is first created. */
    private ProgressBar bar;
    private Button btnUp;
    private Button btnDown;
    private int count = 0;                     //用来计算进度条的值
    private int current= 0;                    //当前进度条的值
    private String show;                       //用来显示消息提示用户
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       requestWindowFeature(Window.FEATURE INDETERMINATE PROGRESS);
                                               //要求视窗为不确定进度条模式
       setContentView(R.layout.main);
       setProgressBarIndeterminateVisibility(true);     //显示不确定进度条

       bar = (ProgressBar) findViewById(R.id.bar);      //获得 View 的操作对象
       btnUp = (Button) findViewById(R.id.btnUp);
       btnDown = (Button) findViewById(R.id.btnDown);

       btnUp.setOnClickListener(new OnClickListener()
        {
             //完成增加进度条的代码
        });
```

```
        btnDown.setOnClickListener(new OnClickListener()
         {
             //完成减少进度条的代码
         });

    }
}
```

到这里整体设计就完成了，注意其中的 Activity.requestWindowFeature(int featureId)方法，该方法用来设置视窗的显示类型，其参数可以设置为 Window.FEATURE_INDETERMINATE_PROGRESS（圆形进度条）或 Window.FEATURE_ PROGRESS（水平进度条），当然还有其他的参数值可以设置，但不属于本小节讨论范围，读者可以自行研究。

3．完成增加进度条功能

这里以 5 为单位增加进度条的进度值：

```
@Override
            public void onClick(View arg0)
            {
                // TODO Auto-generated method stub
                count += 5;
                if (count <= bar.getMax())
                    bar.setProgress(count);                 //设置进度值
                current = bar.getProgress();                //获得进度值
                show = "当前进度为：" + String.valueOf(current);
               Toast.makeText(getBaseContext(), show, Toast.LENGTH_LONG).
                show();
            }
```

使用 ProgressBar.setProgress(int progress)方法完成进度的增加，使用 ProgressBar.getProgress()方法获得当前的进度值。

4．完成减少进度条功能

同样以 5 为单位减少进度条的进度值：

```
@Override
            public void onClick(View arg0)
            {
                //TODO Auto-generated method stub
                count -= 5;
                if (count <= bar.getMax())
                    bar.setProgress(count);                 //设置进度值
                current = bar.getProgress();                //获得进度值
                show = "当前进度为：" + String.valueOf(current);
               Toast.makeText(getBaseContext(), show, Toast.LENGTH LONG).
                show();
            }
```

实现方法与增加相似，这里就不再赘述了。最后的运行效果如图 5.23 所示。

到这里 ProgressBar 的学习就告一段落，希望读者在以后的使用中结合实际写出更加友好的界面，下一小节我们要学习的是拖动条——SeekBar。

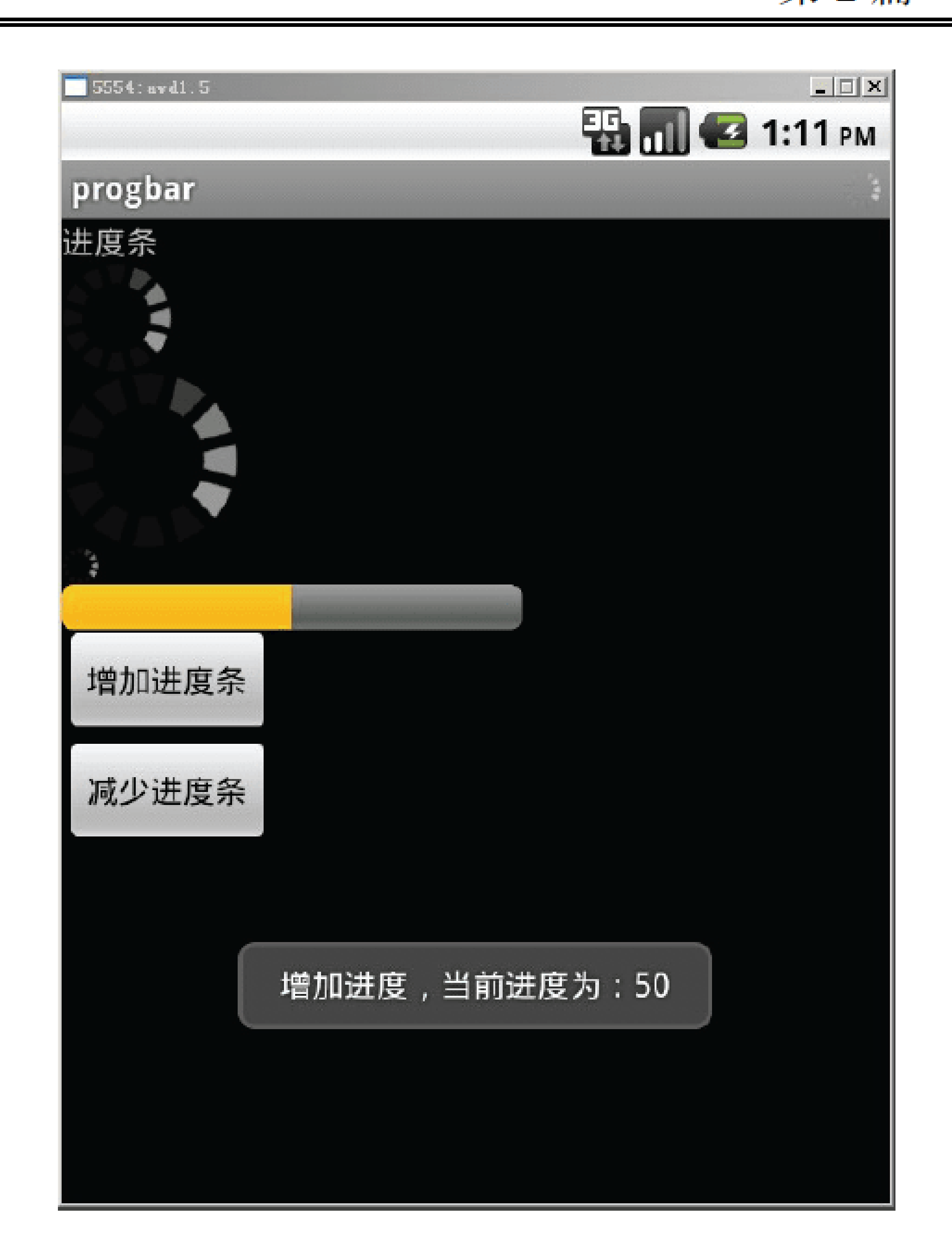




图 5.23　ProgressBar 的使用

5.2.17　使用拖动条——SeekBar

SeekBar 组件的功能与 ProgressBar 类似，不同的是它是可以被拖动的，与用户的交互性更强。

1．xml代码

```
<SeekBar
    android:id="@+id/bar"
     android:layout_width="200dip"
    android:layout_height="wrap_content"
    android:max="100"
 />
```

其 id 设为 bar，宽度设为 200dip，最大值为 100。当然这些属性读者可以自行设置。

2．Java代码

既然 SeekBar 可以被拖动，那必然需要监听被拖动的这个动作，使用 SeekBar.OnSeekBarChangeListener 接口可以完成拖动条改变的监听。使用 SeekBar.setOnSeekBarChangeListener(OnSeekBarChangeListener l)方法可以将监听器与 SeekBar 对象绑定。

实现 OnSeekBarChangeListener 接口时需要完成 3 个方法的重写，分别是：

（1）onStopTrackingTouch(SeekBar arg0)：停止跟踪方法，传递的参数为正在操作的 SeekBar。

（2）onStartTrackingTouch(SeekBar arg0)：开始跟踪方法，传递的参数为正在操作的 SeekBar。

（3）onProgressChanged(SeekBar bar, int current, boolean arg2)：进度改变监听方法，第

一个参数为当前的 SeekBar，第二个参数为当前的进度值，最后一个参数是用户是否正在拖动 SeekBar 的标志，正在操作为 true，否则是 false。

完成了理论学习，我们趁热打铁，把 SeekBar 也给掌握了吧！马上编写一个小例子实践一下到底是个什么样的效果。

5.2.18　实例——简单使用 SeekBar

本实例就只是简单地展现一下 SeekBar 的运行效果以及完成对其拖动的监听。拖动时通过 Toast 提示用户当前正在进行的操作以及当前的进度值。

首先是 xml 代码。

1．xml代码

界面上只有一个 TextView 和一个 SeekBar。SeekBar 的 Id 名为 bar，宽度为 200dip，最大值为 100。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
    <TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="拖动条"
   android:textSize="30sp"
   />
   <SeekBar
       android:id="@+id/bar"
        android:layout_width="200dip"
       android:layout_height="wrap_content"
       android:max="100"
    />
</LinearLayout>
```

接下来分析 Java 部分的代码。

2．Java代码

Java 部分最主要的工作为完成对 SeekBar 的监听，获得当前的进度值，来看代码：

整体设计：

```
package com.wes.seekbar;

import ......                                                      //此处省略部分包
import android.widget.SeekBar;
import android.widget.Toast;
import android.widget.SeekBar.OnSeekBarChangeListener;

public class Seekbar extends Activity {
   /** Called when the activity is first created. */
    private SeekBar bar;
```

```
    private int current= 0;                         //记录当前的进度值
    private String show;                            //需要提示的消息
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        bar = (SeekBar) findViewById(R.id.bar);     //获得 SeekBar 的操作对象
        bar.setOnSeekBarChangeListener(new OnSeekBarChangeListener()
                                                    //设置监听器
         {
             //此处完成对 SeekBar 的监听
         });
    }
}
```

整体的框架非常简单，关键是其中实现 OnSeekBarChangeListener()接口的方法。

3．实现OnSeekBarChangeListener()接口

前文已经提到过实现时需重写的 3 个方法，其执行的顺序为：首先执行 onStartTrackingTouch(SeekBar bar)，然后执行 onProgressChanged(SeekBar bar, int current, boolean arg2)，最后是 onStopTrackingTouch(SeekBar bar)。再具体一些为：当单击滑动块时执行 onStartTrackingTouch，拖动时执行 ProgressChanged，松开滑动块时执行 onStopTrackingTouch。

```
@Override
            public void onStopTrackingTouch(SeekBar bar)
            {
                current = bar.getProgress();            //获得当前进度值
                show = "结束拖动，当前进度为：" + String.valueOf(current);
                Toast.makeText(getBaseContext(), show, Toast.LENGTH LONG).
                show();
            }

            @Override
            public void onStartTrackingTouch(SeekBar bar)
            {
                current = bar.getProgress();            //获得当前进度值
                show = "开始拖动，当前进度为：" + String.valueOf(current);
                Toast.makeText(getBaseContext(), show, Toast.LENGTH_LONG).
                show();
            }

            @Override
            public void onProgressChanged(SeekBar bar, int current, boolean
            arg2)
            {
                show = "进度条改变中…"+String.valueOf(current);
                                                //current 参数即为当前的进度值
                Toast.makeText(getBaseContext(), show, Toast.LENGTH_
                SHORT).show();
            }
```

又一个实例编写完毕了，最后的运行效果如图 5.24 所示。







图 5.24　SeekBar 的使用

本小节到这里就结束了，相信读者应该能完成 ProgressBar 和 SeekBar 的使用了吧。作为两个功能相似的组件，到底选择哪一种，这个问题就是仁者见仁，智者见智了。

5.2.19　使用图片视图——ImageView

本小节将讲解 ImageView，也就是图片视图的使用。使用 ImageView 需要哪些步骤呢？第一步很多读者已经知道了，编写 xml 代码。

1. xml代码

```
<ImageView
 android:id="@+id/imageView1"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:src="@drawable/f1"
/>
```

这里将其 Id 设为 ImageView2，比较特色的是 android:src 属性，设置了该 ImageView 需要显示的内容，这里设置为 drawable 文件夹下的 f1.png 图片。非常简单，这里就不再赘述。接下来看 Java 代码。

2. Java代码

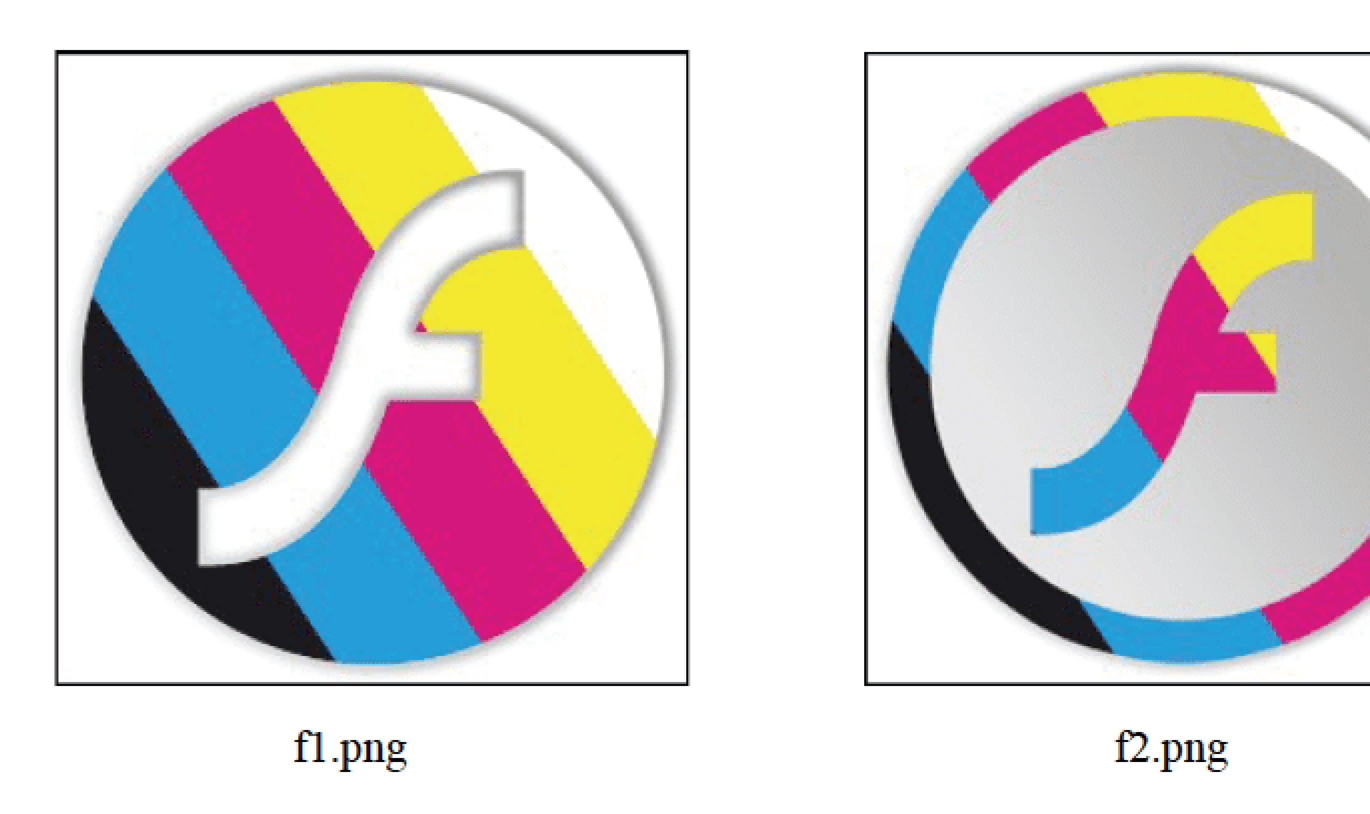
在 Java 代码中同样非常简单，首先通过 Activity.findViewById(int id)方法得到 ImageView 的对象，接着如果你不想做任何改变就完成了 Java 部分代码，如果你希望更改 ImageView 的图片 name 可以使用 ImageView.setImageDrawable(Drawable drawable)方法设置图片内容。当然还有更多的属性可以设置，比如可以通过 ImageView.setAlpha(int alpha)方法设置透明度。更多的属性设置请读者自行参阅 API 文档。

5.2.20　实例——ImageView 的重叠效果

在本实例希望通过 SeekBar 来拖动 ImageView 的位置。通过本实例读者可以更好地掌

握前一小节讲解的 SeekBar 的使用，也可以了解 ImageView 的重叠效果。好了，马上来让 ImageView 跟着你的手指移动吧！

首先准备两张 png 格式的图片，如图 5.25 所示。



f1.png　　f2.png

图 5.25　准备的图片

1. xml代码

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"        //使用绝对布局
    android:layout_height="fill_parent"
    >

    <ImageView                                //第一层的图片视图
         android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="50px"
        android:layout_y="10px"
    />

    <ImageView                                //第二层的图片
     android:id="@+id/imageView2"
        android:layout width="wrap content"
        android:layout height="wrap content"
         android:layout x="50px"
        android:layout y="10px"
    />
    <Button                                   //按钮用于将图片分开或重叠
     android:id="@+id/btn"
        android:layout width="wrap content"
        android:layout height="wrap content"
         android:layout_x="400px"
        android:layout y="500px"
        android:text="分离"
    />
    <SeekBar                                  //拖动条用于控制第一层 ImageView
         android:id="@+id/bar"
        android:layout width="150px"
        android:layout height="wrap content"
         android:layout_x="380px"
        android:layout_y="600px"
```

```
        android:max="100"
    />
</AbsoluteLayout>
```

2. Java代码

首先完成整体设计，得到一些需要操作的 View 的对象，并设置 SeekBar 和 Button 的事件监听。

（1）整体设计

```
package com.wes.imageView;

import ......                                              //省略部分导入包
import android.view.Window;
import android.widget.AbsoluteLayout;                     //导入绝对布局
import android.widget.AbsoluteLayout.LayoutParams;       //导入绝对布局参数
import android.widget.ImageView;                          //导入 ImageView
import android.widget.SeekBar;                            //导入拖动条
import android.widget.SeekBar.OnSeekBarChangeListener; //导入监听器
import android.widget.Toast;                              //导入 Toast 消息提示

public class Imageview extends Activity {
    ImageView imageView1;
    ImageView imageView2;
    Button     button;
    SeekBar    bar;
     LayoutParams params1;
    LayoutParams params2;
    boolean isSeparated = false;                       //两张图片是否分离的标志

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);  //去掉标题
        setContentView(R.layout.main);

       //得到布局中的组件的对象
        button      = (Button)     findViewById(R.id.btn);
        bar         = (SeekBar)   findViewById(R.id.bar);
        imageView1 = (ImageView) findViewById(R.id.imageView1);
        imageView2 = (ImageView) findViewById(R.id.imageView2);

        //为 ImageView 设置图片
        imageView1.setImageDrawable(getResources().getDrawable(R.
        drawable.f1));
        imageView2.setImageDrawable(getResources().getDrawable(R.
        drawable.f2));

        //得到 ImageView 的布局参数
        params1 = (LayoutParams) imageView1.getLayoutParams();
        params2 = (LayoutParams) imageView2.getLayoutParams();

        //为 SeekBar 设置监听器
        bar.setOnSeekBarChangeListener(new OnSeekBarChangeListener()
         {
              //此处完成拖动条的监听
         });
        //为按钮设置监听器
```

```
        button.setOnClickListener(new OnClickListener()
         {
             //此处完成按钮的监听
         });
    }
}
```

（2）拖动条的事件监听

这里根据 SeekBar 的值确定 ImageView 的 layout_x 的位置，layout_y 不变，这样就实现了图片的横向移动。

```
@Override
            public void onStopTrackingTouch(SeekBar arg0)
            {
            }

            @Override
            public void onStartTrackingTouch(SeekBar arg0)
            {
            }

            @Override
            public void onProgressChanged(SeekBar arg0, int position,
            boolean arg2)
            {
                params1.x = position;
                            //根据 SeekBar 的 position 决定 ImageView1 的位置
                ImageView1.setLayoutParams(params1);
                            //为 ImageView 设置参数
            }
```

（3）按钮的事件监听

这里实现的功能是：如果图片分离则显示“重叠”按钮，单击后两张图片重叠，如果重叠则显示“分离”按钮，单击后图片分离。

```
@SuppressWarnings("deprecation")
            @Override
            public void onClick(View arg0)
            {
                if (!isSeparated)                    //如果不分离就使图片分离
                {
                    params1.x = 10;
                    params1.y = 0;
                    imageView1.setLayoutParams(params1);

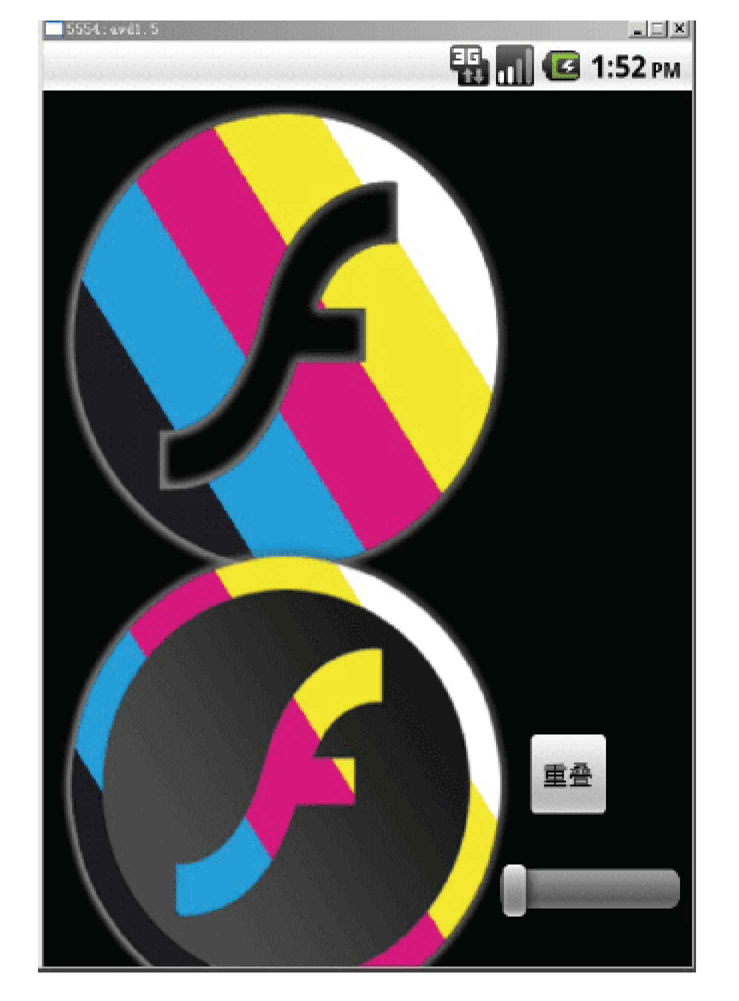
                    params2.x = 10;
                    params2.y = 350;
                    imageView2.setLayoutParams(params2);

                    button.setText("重叠");           //修改 Button 的显示
                    isSeparated = true;              //修改标签为“分离”
                }
                else
                {                                    //如果分离就使其重叠
                    params1.x = 50;
                    params1.y = 50;
                    imageView1.setLayoutParams(params1);
```

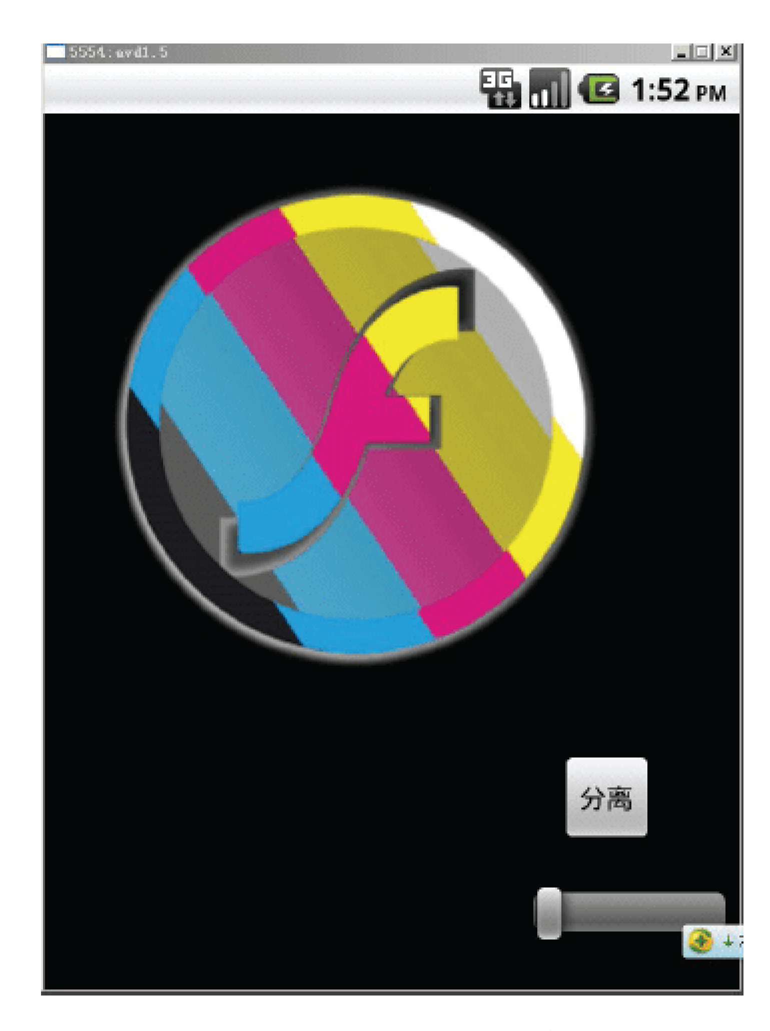
```
                    params2.x = 50;
                    params2.y = 50;
                    imageView2.setLayoutParams(params2);

                    button.setText("分离");                //修改 Button 的显示
                    isSeparated = false;                  //修改标签为“分离”
                }
            }
```

最后的运行效果如图 5.26 所示。



单击“分离”后的效果



单击“重叠”后的效果



拖动拖动条后的效果

图 5.26　运行 ImageView 运行效果图

5.2.21　使用网格视图——GridView

相信大家都用过 Android 操作系统，操作系统中很多的应用按网格排列，一目了然，非常友好。本小节就尝试使用 GridView 完成该类界面的设计。要使用 GridView 网格视图需使用<GridView>标签。其 xml 代码如下：

1. xml代码

```
 <GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/grid"
    android:layout width="match parent"
android:layout height="match parent"
   android:padding="10dp"
   android:verticalSpacing="10dp"
   android:horizontalSpacing="10dp"
   android:numColumns="auto_fit"
   android:columnWidth="60dp"
   android:stretchMode="columnWidth"
   android:gravity="center"
/>
```

接下来对其中的一些属性做出说明：

- ❑ android:padding="10dp"：四边的补丁宽度，这里设置为 10dp。

- android:verticalSpacing="10dp"：垂向间隔，这里设置为 10dp。
- android:horizontalSpacing="10dp"：水平间隔，这里设置为 10dp。
- android:numColumns="auto_fit"：列数，这里设置为自适应。
- android:columnWidth="60dp"：列宽度，这里设置为 60dp。
- android:stretchMode="columnWidth"：缩放模式，这里设置为缩放与列宽大小同步。
- android:gravity="center"：重力，这里设置为居中，类似于 word 文档的居中功能。

2. Java代码

使用 GridView 在 Java 代码中需要进行 4 步：

（1）新建 GridView 对象。

（2）新建适配器。

（3）为 GridView 设置适配器。

（4）为 GridView 设置监听器。

接下来仔细分析每一步的实现和功能：

（1）新建 GridView 对象。

通过 Activity.findViewById(int id)方法获得 GridView 对象，对于现在的读者朋友们已经不是问题了吧。

（2）新建适配器。

为了达到更好的效果，我们需要自己写一个继承自 BaseAdapter 的适配器类，实现其中的一些函数，如 getView(int position)等。这在之后的实例中会有详细的讲解，这里只做简单介绍。

（3）为 GridView 设置适配器。

使用 GridView.setAdapter(ListAdapter adapter)设置适配器。这一步非常简单，但必不可少，没有这一步，我们的 Adapter 和 GridView 就无法关联。

（4）为 GridView 设置监听器。

使用 AdapterView.setOnItemClickListener(OnItemClickListener listener)设置监听器，这在本章中已经讲解过，这里就不再详细叙述。

5.2.22　实例——通过宫格视图展示相应的应用

本例模仿 Android 系统的界面，使用 GridView 显示系统中安装的开机自动启动的应用。本例中将结合 GridView 和 ImageView 达到预期效果，并将首次自己完成适配器的代码编写。注意：适配器的编写非常重要，这在很多 Widget 中都需要使用。首先编写 xml 代码。

1. xml代码

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <GridView xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:id="@+id/grid"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
   android:padding="10dp"
   android:verticalSpacing="10dp"

   android:horizontalSpacing="10dp"
   android:numColumns="auto_fit"
   android:columnWidth="60dp"
   android:stretchMode="columnWidth"

   android:gravity="center"
   />
</LinearLayout>
```

我们看到，在线性布局中我们只是加入了一个 GridView 组件，其各类属性的意义已经在上一小节中讲解，这里就不再展开，读者如仍有疑问，请翻阅上一小节。

2. Java代码

本节的 Java 代码包含两个类，一个是 Activity 类，另一个是 Adapter 适配器类。首先我们观察 Activity 类：

（1）Gridview 类

a. 整体设计：

```
package com.wes.gridview;

import ...                                          //省略部分导入包
import android.content.pm.ResolveInfo;
                                                    //导入包管理器下的信息类
import android.widget.AdapterView.OnItemClickListener;     //导入监听器
import android.widget.GridView;                     //导入网格视图类
public class Gridview extends Activity {
    private GridView grid;                          //声明 GridView 的对象
    private List<ResolveInfo> apps;                 //该列表用于存放查询后的应用信息
    private MyAdapter adapter;                      //适配器对象
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       apps = queryApps();                                 //查询需要的应用
       setContentView(R.layout.main);
       grid = (GridView) findViewById(R.id.grid);          //实例化 grid 对象

       adapter = new MyAdapter(this, apps);                //新建 Adapter 对象
       grid.setAdapter(adapter);                           //设置适配器
       grid.setOnItemClickListener(new OnItemClickListener()
        {                                                  //设置监听器
            //此处完成监听器
        });
   }
```

这里的第一步是查询我们需要展现的应用，存放到列表中，这样我们就有了数据源。接下来的 4 个步骤就是上一小节中我们讲解的 4 个步骤了，简单明了。接下来完成监听器的功能实现。

b. 实现监听器功能

监听被单击的选项，显示该应用的名称：

```
@Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int
            position, long arg3)
            {
                // TODO Auto-generated method stub
                ResolveInfo info = apps.get(position);
                                            //得到单击的 ResolveInfo 对象
                String name = info.activityInfo.loadLabel(getPackage
                Manager()).toString();      //得到应用的名字
               Toast.makeText(getBaseContext(), name, Toast.LENGTH_LONG).
                show();                     //显示名字
            }
```

c. 实现查询应用功能

这里使用了 PackageManager.queryIntentActivities(Intent arg0, int arg1)方法查询相应的 Activitys。

通过该方法可以得到符合 Intent 的 Activity 列表。这里的 Intent 参数需要设置两个属性：一个是 Action——动作，一个是 Category——种类。

```
         private List<ResolveInfo> queryApps() {
        Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
                                            //设置 Action 属性为 MAIN
        mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);
                                            //设置 Category 属性为 LAUNCHER
        apps = getPackageManager().queryIntentActivities(mainIntent, 0);
                                            //执行查询
        return apps;
}
```

本例中的 Action 值设为 MAIN，即主 Activity；Category 设为 LAUNCHER，即开机自启动。

（2）MyAdapter 类

本例没有使用系统提供的 ListAdapter 或 SimpleAdapter 等，而是自己通过继承 BaseAdapter 编写了自己的 MyAdapter 类。接下来就看一下实现的过程吧！完成自定义 Adapter 需重写 4 个函数，分别是：

- getCount()方法：得到 View 的个数，代码如下：

```
@Override
    public int getCount()
    {
        return apps.size();             //得到个数
    }
```

- getItem(int position)：得到位于 position 的选项数据，代码如下：

```
@Override
    public Object getItem(int position)
    {
        return apps.get(position);      //得到该位置的 Item
    }
```

❑ getItemId(int position)方法：得到 Item 的 Id，也就是其位置了，代码如下：

```
@Override
    public long getItemId(int position)
    {
        return position;                    //得到 Item 的位置
    }
```

❑ getView(int position, View convertView, ViewGroup parent)方法：根据位置得到视图，代码如下：

```
@Override
    public View getView(int position, View convertView, ViewGroup parent)
    {                                       //得到视图
        ImageView imageView;
        if (convertView == null)
        {
            imageView = (ImageView) new ImageView(context);
                                            //实例化一个 ImageView
            imageView.setScaleType(ImageView.ScaleType.FIT CENTER);
                                            //设置尺度模式，缩放以处于中间
            imageView.setLayoutParams(new GridView.LayoutParams(50, 50));
                                            //设置布局参数
        }
        else
        {
            imageView = (ImageView) convertView;   //直接返回 converView
        }
        ResolveInfo info = apps.get(position); //得到该位置的 ResolvsInfo
        //为 ImageView 设置背景图片
        imageView.setImageDrawable(info.activityInfo.loadIcon(context.
        getPackageManager()));
        return imageView;
    }
```

这里首先判断传入的 convertView 参数是否为空，如果是空就新建一个 ImageView 返回，如果不是空则返回该 convertView。

3. 最后的整体代码

如下所示：

```
package com.wes.gridview;

import ...
import android.content.pm.ResolveInfo;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class MyAdapter extends BaseAdapter
{
    Context context;                    //上下文关系
    List<ResolveInfo> apps;             //数据源
    public MyAdapter(Context ctx,List<ResolveInfo> apps)
    {                                   //构造函数
```

```
        this.context = ctx;
        this.apps = apps;
    }
    @Override
    public int getCount()
    {
        return apps.size();             //得到个数
    }

    @Override
    public Object getItem(int position)
    {
        return apps.get(position);      //得到该位置的 Item
    }

    @Override
    public long getItemId(int position)
    {
        return position;                //得到 Item 的位置
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent)
    {                                   //得到视图
        ImageView imageView;
        if (convertView == null)
        {
            imageView = (ImageView) new ImageView(context);
                                        //实例化一个 ImageView
            imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
                                        //设置尺度模式，缩放以处于中间
            imageView.setLayoutParams(new GridView.LayoutParams(50, 50));
                                        //设置布局参数
        }
        else
        {
            imageView = (ImageView) convertView;   //直接返回 converView
        }
        ResolveInfo info = apps.get(position); //得到该位置的 ResolvsInfo
        //为 ImageView 设置背景图片
        imageView.setImageDrawable(info.activityInfo.loadIcon(context.
        getPackageManager()));
        return imageView;
    }

}
```

到这里，代码就编写完成了，运行后效果如图 5.27 所示。

关于 GridView 的讲解就先到这里了，下一小节我们将讲解的是 Toast 的使用。

5.2.23　使用消息提醒——Toast

Toast，原意是“吐司”，也就是一种面包。在 Android 中，它表示一种消息机制，没有焦点，显示一段时间后自动消失。至于这种消息提醒机制叫做 Toast，笔者却以为是因其展现形式类似于一个切片面包。实际上，我们在之前的实例中已经大量使用了 Toast，也许

细心的读者已经发现并学会了初步的使用，本小节将对其进行一个比较深入的讲解。



初识化界面



单击警徽标志后

图 5.27　GridView 效果图

使用 Toast 大致分为几种类型：

1. 默认的Toast

这也是笔者之前一直在使用的 Toast 使用方法，该 Toast 会将文字显示在屏幕的底部，其方法为：

```
Toast.makeText(Context context, CharSequence text, int duration)
```

这里需要的 3 个参数意义如下：

- ❑ Context context ：上下文关系，这在很多方法中都需要使用，相信大家应该理解了。
- ❑ CharSequence text：这个参数就是你希望显示的内容了，在这里填上一个字符串，如“您有新的消息”。
- ❑ int duration：duration 的意义是持续时间，int 型并不是需要你填写整数，而是填写 Toast.LENGTH_LONG（大约 5 秒），长时间显示；或者填写 Toast.LENGTH_SHORT（大约 3 秒），短时间显示。

当然，在调用完这个方法以后，千万别忘记调用：

```
Toast.show()
```

只有调用了该方法后，Toast 才会正常显示出来，如果不调用该 show()方法，之前的一切就是无用功了。

2. 调整位置显示的Toast

Toast 如果只是在屏幕底部出现，很多时候就觉得不是很方便，也显得比较单调，这个时候我们可以使用方法：

```
Toast.setGravity(int gravity, int xOffset, int yOffset)
```

设置重力，这在之前的一些组件中也有使用，这里的 3 个参数分别为重力、x 轴偏移、y 轴偏移，一幕了然。第一个参数 gravity 可以设置的选择有很多，比如：Gravity.CENTER、Gravity.TOP、Gravity.RIGHT 等等。至于使用的效果就由读者自己去尝试了，本书篇幅有限就不再一一讲解了。

3. 显示图片的Toast

很多读者也许会说，只是显示文字太不过瘾了，我还希望显示图片，可以吗？当然可以！如果需要显示图片，只需调用方法：

```
Toast.setView(View view)
```

读者看到这里的参数很简单，一个被实例化的 View 就可以了。也就是说，不仅仅可以显示图片，你甚至可以显示任何你希望显示的视图，一个 ImageView、一个 TextView、一个 Button 甚至自己写一个 LinearLayout 都可以作为参数被 Toast 显示。

4. 同时显示文字和图片的Toast

事实上，该种 Toast 使用前一种的方法同样可以实现。只需创建一个 LinearLayout 里面包裹一个 TextView 和一个 ImageView 就可以了。这里介绍的是一个较为方便的方法，共需要 3 个步骤：

（1）新建一个默认的 Toast，这在本节已经有过讲解，这里就不再赘述。

（2）通过 Toast.getView()方法获得该 Toast 的 View，并将之转换为 LinearLayout 布局。

（3）通过 ViewGroup.addView(View child, int index)方法将 ImageView 添加入布局中，这里的第一个参数就是要添加的 View，第二个参数是显示的位置，如设置为 0 就显示在文字上方，设置为 1 就显示在文字下方。注意这里的 LinearLayout 默认是垂直布局的。

5.2.24　实例——Toast 的 4 种实现

本例将继续使用 5.2.20 小节中的实例，还记得那个 ImageView 的使用吗？在那个实例中我们结合使用了 SeekBar 和 ImageView。本小节我们将重新编写 OnSeekBarChangeListener()的实现方法，其他部分的代码不变。那么接下来我们就开始工作吧！

首先回顾一下 SeekBar 的监听事件的监听，使用方法：

```
//为 SeekBar 设置监听器
       bar.setOnSeekBarChangeListener(new OnSeekBarChangeListener()
        {
            //此处完成拖动条的监听
        });
```

在其中的实现为：

```
@Override
            public void onStopTrackingTouch(SeekBar arg0)
            {
            }

            @Override
            public void onStartTrackingTouch(SeekBar arg0)
```

```
        {
        }

        @Override
        public void onProgressChanged(SeekBar arg0, int position,
        boolean arg2)
        {
        }
```

首先，重写 onStartTrackingTouch()方法，在开始滑动时，使用默认的 Toast 提醒用户，滑动开始了：

```
@Override
        public void onStartTrackingTouch(SeekBar arg0)
        {
            Toast toast = Toast.makeText(getBaseContext(), "请开始拖动,
            将两个图片重叠", Toast.LENGTH_SHORT);
            toast.show();
        }
```

Toast 默认的使用方法读者务必做到驾轻就熟。

接着，重写 onProgressChanged()方法，在滑动过程中通过 SeekBar 的值改变 ImageView 的位置：

```
public void onProgressChanged(SeekBar arg0, int position, boolean arg2)
        {
            params1.x = position;
                        // 根据 SeekBar 的 position 决定 ImageView1 的位置
            imageView1.setLayoutParams(params1);
                        //为 ImageView 设置参数
        }
```

最后，重写 onStopTrackingTouch()方法，在滑动结束时判断 ImageView 的具体位置。如果两个图片重叠就同时显示文字和图片，恭喜用户完成重叠；如果偏左了，则通过改变位置的 Toast 通知用户偏左了；如果偏右了，则通过只显示图片的 Toast 通知用户，不显示文字。代码如下：

```
@Override
        public void onStopTrackingTouch(SeekBar arg0)
        {
            if (params1.x <= 55 && params1.x >= 45)//该区域内判断为重叠
            {
Toast toast = Toast.makeText(getBaseContext(), "恭喜你，成功重叠！",
Toast.LENGTH SHORT);
                LinearLayout toastLayout = (LinearLayout) toast.
                getView();              //获得布局
                ImageView imageView = new ImageView(getBaseContext());
                                        //获得图片视图
               imageView.setImageDrawable(getResources().getDrawable
                (R.drawable.doolar));
               imageView.setAlpha(100);     //设置透明度，参数为 0 则不显示

               toastLayout.addView(imageView,0);        //添加图片视图
               toast.setDuration(Toast.LENGTH_LONG);    //设置持续时间
               toast.show();                            //显示 Toast
            }
            else if(params1.x <= 45)                  //该区域内判断为偏左
```

```
            {
                Toast toast = Toast.makeText(getBaseContext(), "呀！偏
                左了。。", Toast.LENGTH_SHORT);
                toast.setGravity(Gravity.CENTER, 0, 0);
                                                    //设置显示在中间
                toast.show();
            }
            else if (params1.x >= 55)               //该区域内判断为偏右
            {
                ImageView imageView = new ImageView(getBaseContext());
              imageView.setImageDrawable(getResources().getDrawable
              (R.drawable.doolar));
              Toast toast = new Toast(getBaseContext());

              toast.setView(imageView);         //设置 Toast 要显示的视图
              toast.setDuration(Toast.LENGTH LONG);
              toast.show();
            }
        }
```

最后运行效果如图 5.28 所示。



默认的 Toast



改变位置的 Toast



只显示图片的 Toast



同时显示文字和图片的 Toast

图 5.28　Toast 的各种使用

5.3　使用列表视图（ListView&ExpandableListView）

通过了 5.2 节的学习，我们已经学会使用了很多平时开发中需要使用的组件，本节中要讲解的是列表视图。通过列表视图可以方便地列出一系列信息，开发中我们经常使用，非常重要。本节将学习其使用方法，希望读者能很好地掌握并使用。

5.3.1　使用列表——ListView

现在，想象一下，你希望编写一个程序，用以显示你即将去超市要购买的物品，你准备怎么办？通过我们已经学习的知识，完全可以做到这一点：首先，在脑子里要把准备买的东西想好；然后根据买的物品的个数编写相应个数的 TextView；接着将所有的 TextView 添加到一个 LinearLayout 中，并设置其垂直显示；最后将每个 TextView 的内容一一设置为对应的内容。

如果你能这样做，我会很欣赏，因为你有思路，有耐心，有毅力。但是，这样做真的好吗？作为一个开发人员我们深知效率是编程过程中很重要的一个部分。如果我们傻傻地进行这样的重复劳动，是不是太没有技术含量了？这个时候，我们就可以使用 ListView 啦！

使用 ListView 编写列表，可以帮助我们从单调重复的工作中解脱出来，接下来我们就来学习使用列表视图——ListView，首先编写其 xml 代码。

1. xml代码

xml 代码中分为两个部分，第一个是 ListView 的 xml 代码：

```
<ListView
    android:id="@+id/lv"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>
```

第二个是每个子项的视图的 xml 代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
  <TextView
    android:id="@+id/name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

该 xml 文件即是每个具体子项的布局文件了，在该布局中可以添加任何你希望完成的效果。当然不能忘记有一个 TextView，这个是必须。该 TextView 用来显示数据源中的数据。

接下来是 Java 部分的代码。

2. Java代码

Java 部分需要完成的步骤如下：

（1）使用 Activity.findViewById(int id)方法得到 ListView 的对象。

（2）将原始数据转换为适配器需要的数据结构，这一点非常重要，也是使用 ListView 的一个难点，在之后的实例中会有具体的讲解。

（3）新建适配器对象。这里可供选择的方法有很多，可以使用自定义的 Adapter，当然很多时候为了简单，我们可以使用 SimpleAdapter 类。具体构造方法为：

```
SimpleAdapter(Context context,
            List<? extends Map<String, ?>> data,
int resource,
String[] from,
int[] to)
```

这里有 5 个参数：

第一个参数是上下文关系，我们不再管它。

第二个参数是数据源，需要的数据结构是一个 List<>里面的对象应该继承自 Map(Sting,?)。指定其“键”必须是 String 类型，事实上，“值”我们一般也使用 String 类型。这个就是我们为什么要进行第二个步骤的原因所在了。

第三个参数是资源 Id，也就是每个子项的布局文件。

第四个参数是 data 数据源中的“键”(key)，通过该键可以得到 data 中需要展示的“值”，也就是 value。

第五个参数是 item 布局文件中的 TextView 的 Id，也就是数据要显示的具体地方。

（4）为 ListView 设置适配器。使用方法：

```
ListView.setAdapter(ListAdapter adapter)
```

接下来，依然通过一个实例学习使用列表视图——ListView。

5.3.2　通过实例学习列表

本例学习使用 ListView 罗列出 Android 的 SDK 的一些版本，每个项目希望同时显示 Android 的图标和版本号。首先我们编写 xml 代码。

1. xml代码

分为两个 xml 文件，首先是 main.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<ListView                                                              //列表
     android:id="@+id/lv"
     android:layout_width="wrap_content"
```

```
    android:layout height="wrap content"
/>

</LinearLayout>
```

接着需编写每个子项的布局文件，这里将之命名为 item.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"                    //水平布局
   android:layout width="fill parent"
   android:layout_height="fill_parent"
   >
   <ImageView                                          //图片视图
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:src="@drawable/icon"                       //资源为 Android 图标
   />
  <TextView
    android:id="@+id/name"                             //Id 为 name/droidyout
    android:layout_width="wrap_content"
    android:layout height="wrap content"
    android:textSize="20sp"                            //字体大小 20sp
    android:text=""                                    //默认没有内容
    />
</LinearLayout>
```

布局文件非常简单，一个 LinearLayout 中包含一个图片视图和一个文本框，水平布局。

2. Java代码

在 Java 部分需要完成的步骤，上一小节中已经有了详细的讲解，希望读者可以按部就班的独立完成。首先是整体设计部分：

```
package com.wes.list;

import ...                                                     //省略部分导入包
mport java.util.ArrayList;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.Toast;

public class SimpleList extends Activity {

    private String[]                             names;        //数据源
    private ArrayList<HashMap<String, String>>  listItem;    //需求的数据结构
    private ListView                             mListview ; //列表对象
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       initCtrl();                                             //初始化组件

       mListview.setOnItemClickListener((OnItemClickListener) new OnItem
       ClickListener()                                         //设置监听事件
        {
            @Override
```

```
            public void onItemClick(AdapterView<?> arg0, View arg1, int
            position, long arg3)
            {
                Toast.makeText(getBaseContext(), "您选择了" + names
                [position], Toast.LENGTH_LONG).show();
                                                   //显示被单击的选项的内容
            }
        });
    }
```

整体设计中我们将需要进行的工作都放在了 initCtrl()方法中，接下来我们编写 initCtrl()方法。

3．实现初始化组件

在该函数中需要按照之前的讲解完成 4 个步骤，读者应该了然于心了：

```
    protected void initCtrl()
    {
    mListview = (ListView) findViewById(R.id.lv);       //获得 listView 对象
    listItem = loadData();                              //加载数据
    SimpleAdapter listItemAdapter = new SimpleAdapter(getBaseContext(),
                                                        //获得适配器
                                                    listItem,
                                                    R.layout.item,
                                                    new String[]{"ItemName"},
                                                    new int[] {R.id.name});
        mListview.setAdapter(listItemAdapter);          //设置适配器
    }
```

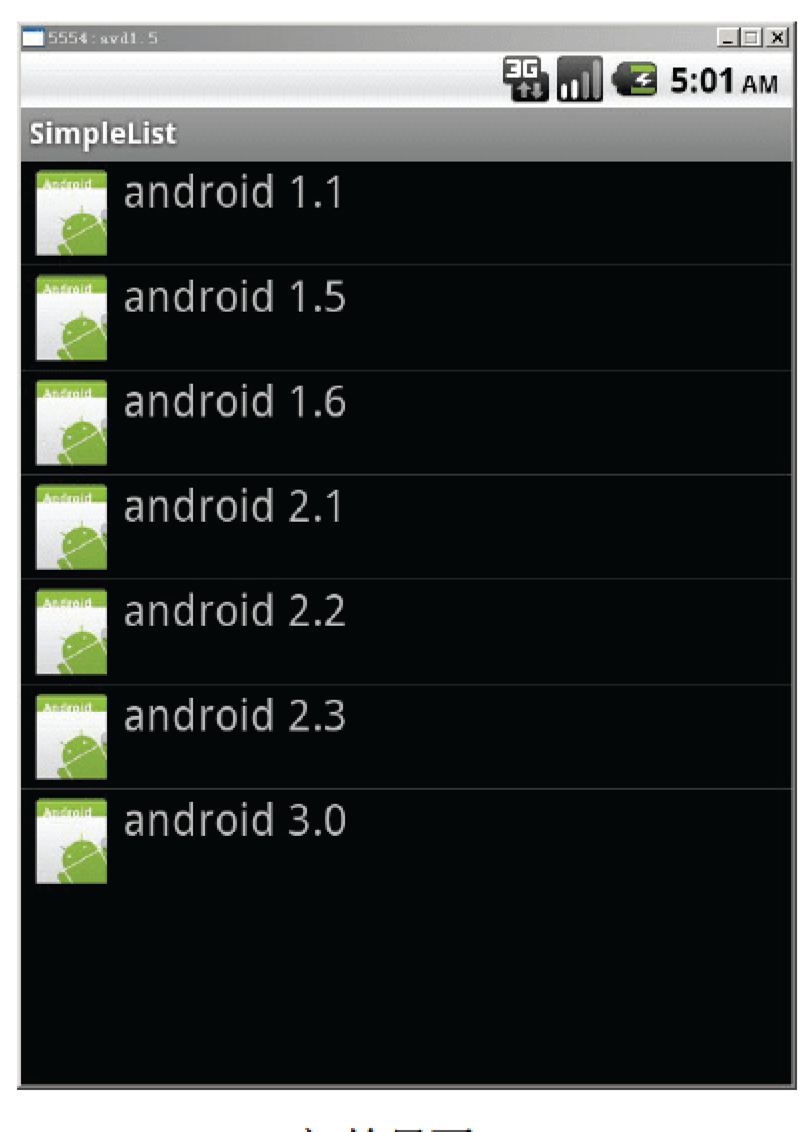
在该方法中加载数据依然是重头戏，接下来就完成加载数据的方法。

4．实现加载数据的方法

首先我们使用 String 数组提供原始数据，接着遍历数组，将每个数据都赋予一个 Key，以便 adapter 可以很方便地得到，最后将每个键值对都添加到 List<>中。

```
    private ArrayList<HashMap<String, String>> loadData()
    {
    names = new String[]{"android 1.1","android 1.5","android 1.6","android
    2.1","android 2.2","android 2.3","android 3.0"};
    listItem = new ArrayList<HashMap<String, String>>();
                                                  //最后需要的数据结构
        for(int i = 0;i < names.length; i++)      //遍历数组
        {
            HashMap<String, String> map = new HashMap<String, String>();
            String name = names[i];
            map.put("ItemName", name);            //以键值对的形式保存
            listItem.add(map);                    //将 HashMap 添加到 list 中
        }
        return listItem;
    }
}
```

经过该方法的转换，这就是 adapter 需要的数据结构了。最后运行效果如图 5.29 所示。



初始界面



单击 android 2.2 后

图 5.29　ListView 的使用

5.3.3　使用可扩展列表——ExpandableListView

将数据进行一个简单的罗列只是一个简单的 ListView 的使用。感觉方便是方便，但功能还不够强大，如我们平常使用的 QQ，首先显示一层列表，接着单击某个选项继续弹出下一层列表，感觉很方便、很实用。那这个效果是怎样编写出来的呢？这就是本小节要讲解的可扩展列表了。可扩展列表包含两层，一层是 GroupLayout，也就是通常意义上的第一层列表；第二层是 ChildLayout，也就是第二层列表。

使用 ExpandableListView 首先编写 xml 代码。

1. xml代码

ExpandableListView 的 xml 代码：

```
<ExpandableListView
 android:id="@id/android:list"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:drawSelectorOnTop="false"
 />
```

注意这里的 id 使用的是 android:list，这是为了在继承 ExpandableListActivity 时可以被识别。不能修改为其他 id，否则会出现异常。

接着还需编写 group 的布局文件，布局文件中必须包含：

```
<TextView
   android:id="@+id/groupto"
   android:layout width="fill parent"
   android:layout height="fill parent"
/>
```

这里的 id 是可以自己指定的，在新建 Adapter 对象时需使用。

接着还需编写 child 的布局文件，同样一定要包含：

```
<TextView
    android:id="@+id/childto"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
 />
```

同样只是一个 TextView 用以显示数据。接下来需要编写 Java 代码。

2. Java代码

完成 ExpandableListView 的使用与 ListView 类似，需要 4 个步骤：

（1）获得 ExpandableListView 的操作对象，这一步笔者就不再赘述了。

（2）组装需要提供给适配器的数据源，这一步同样是重点，会在之后详细讲解。

（3）新建适配器对象，这里同样有多种选择，首先学习 SimpleExpandableListAdapter，使用方法如下：

```
new SimpleExpandableListAdapter(
                Context context,                        //context
                List<? extends Map<String, ?>> groupData,  //一级条目的数据
                int groupLayout,               //用来设置一级条目样式的布局文件
                String[] groupFrom,            //指定一级条目数据的 key
                int[] groupTo,                 //指定一级条目数据显示控件的 id
                List<? extends List<? extends Map<String, ?>>> childData,
                                               //指定二级条目的数据
                , int childLayout,             //用来设置二级条目样式的布局文件
                String[] childFrom,            //指定二级条目数据的 key
                int[] childTo);                //指定二级条目数据显示控件的 id
```

这里的参数较多，第一眼看上去很可怕，但理清思路后就发现，这些参数都是有迹可循的。我们发现这里有 9 个参数，第一个参数是上下文关系我们暂时不去管它。接下来就是 8 个参数，我们将其分为两部分，groupData、groupLayout、groupFrom、groupTo 为一级条目提供需要的参数，childData、childLayout、childFrom、childTo 为二级条目提供参数。实际上我们只要学会其中的一部分，另一部分自然就会了。而每部分的 4 个参数和 ListView 的 Adapter 的参数类似，意义也相同。这里笔者就不再一一解释了，如有疑问，请查阅上一小节。

这个时候我们发现：SimpleExpandableListAdapter 经过抽丝剥茧的一番分析，可以还原为 SimpleAdapter 的编写，也就应了一个成语“殊途同归”。

（4）为 ExpandableListView 设置适配器，这一步读者应该已经驾轻就熟了。

学习完理论知识以后，马上编写一个小小的实例来实践一下吧！

5.3.4　实例——简单使用 ExpandableListView

本例只是为了实践一下可扩展列表的使用，因为毕竟只是理论讲解读者可能在编写实例时有一定困难。故本例依然带有讲解的性质，而非实际应用。

依照上一小节的讲解，首先编写 3 个 xml 文件，笔者分别将之命名为：

1．main.xml代码

在线性布局中需要添加 ExpandableListView 组件，注意其 id 的命名。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
    <ExpandableListView
     android:id="@id/android:list"
     android:layout width="fill parent"
     android:layout height="fill parent"
     android:drawSelectorOnTop="false"
     />
</LinearLayout>
```

2．group.xml代码

在线性布局中添加一个文本框，其 id 为 groupto。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
  <TextView
     android:id="@+id/groupto"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:paddingTop="10px"                    //距离上面的组件 10px
     android:paddingLeft="60px"                   //距离左边的组件 60px
     android:paddingBottom="10px"                 //距离底部的组件 10px
     android:textSize="26sp"                      //字体大小为 26sp
     android:text="No data"
     />
</LinearLayout>
```

3．child.xml代码

布局文件与 group.xml 类似，这里就不再列出，注意将其 id 改为 childto。

4．Java代码

首先是整体设计，按照之前的讲解共分为 4 个步骤，但是这里读者会发现这只有 3 个步骤了，为什么呢？因为我们继承了 ExpandableListAcitivty，父类已经帮我们完成了 ExpandableListView 的实例化，这也是为什么 ExpandableListView 的 id 不能改变的原因，修改之后 ExpandableListAcitivty 就无法找到它了。

5．整体设计

```
package com.test;

import ...                                        //省略部分导入包
```

```
import android.app.ExpandableListActivity;
import android.widget.SimpleExpandableListAdapter;

//创建一个 Activity，继承 ExpandableListAcitivty
public class MainActivity extends ExpandableListActivity {
    List<Map<String, String>> groups;  //定义一个 List，存放所有的一级标题的数据
    List<List<Map<String, String>>> childs;
                              //定义一个 List，存放所有的二级标题需要的数据
    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        loadGroupData();                       //加载一级条目需要的数据
        loadChildData();                       //加载二级条目需要的数据

        //新建一个 SimpleExpandableListAdapter 对象
        SimpleExpandableListAdapter sela = new SimpleExpandableList
        Adapter(
                this,                          //context
                groups,                        //一级条目的数据
                R.layout.group,                //用来设置一级条目样式的布局文件
                new String[] { "group" },      //指定一级条目数据的 key
                new int[] { R.id.groupto },//指定一级条目数据显示控件的 id
                childs,                    //指定二级条目的数据
                R.layout.child,            //用来设置二级条目样式的布局文件
                new String[] { "child" },  //指定二级条目数据的 key
                new int[] { R.id.childto });//指定二级条目数据显示控件的 id

        //为当前的 ExpandableListActivity 设置适配器
        setListAdapter(sela);
    }
```

3 个步骤非常简单，思路很清晰，接下来就是完成以及条目数据和二级条目数据的加载了。

6. 完成加载一级条目数据

其需要的数据结构与 SimpleAdapter 相同，每个需要显示的数据都要添加到一个 Map<String,String>中，最后每个 Map 都要加入到一个 List<Map<String,String>>中。

```
    public void loadGroupData()
    {
        groups = new ArrayList<Map<String, String>>();
        //向一级标题添加数据
        Map<String, String> group1 = new HashMap<String, String>();
        group1.put("group", "group1");
        Map<String, String> group2 = new HashMap<String, String>();
        group2.put("group", "group2");
        groups.add(group1);
        groups.add(group2);
    }
```

7. 完成加载二级条目的数据

每个一级条目都对应着一组数据，也就是一个列表。接着每个列表中都存放若干个键

值对，而这些键值对的值才是我们最后需要显示的数据。

所以该方法我们可以将之看成两步：

第一步：为每个一级条目新建一个 List，在其中存放若干键值对数据。

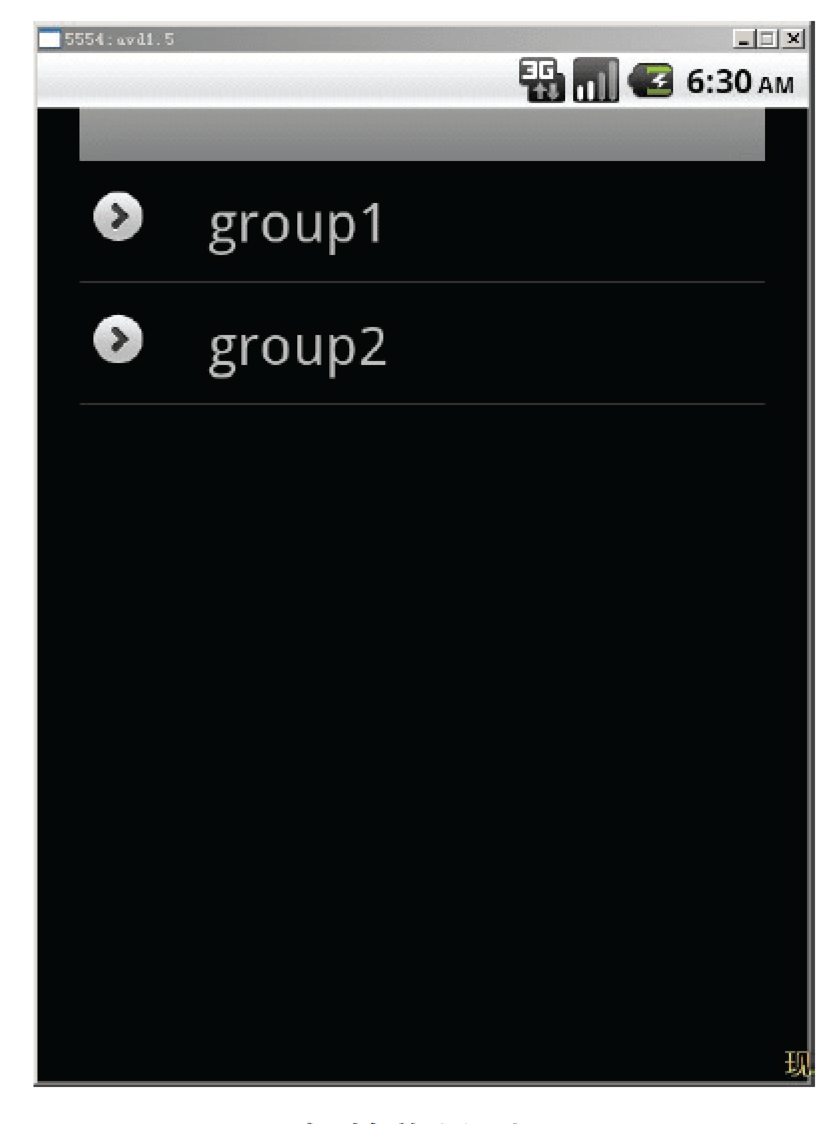
第二部：将每个新建的 List 添加到总的 List 中，以提供给 Adapter 解析。

```
    public void loadChildData()
    {
        childs = new ArrayList<List<Map<String, String>>>();
        //定义一个 List，存放第一个一级标题需要的二级标题的数据
        List<Map<String, String>> child1 = new ArrayList<Map<String,
        String>>();
        Map<String, String> child1Data1 = new HashMap<String, String>();
        child1Data1.put("child", "child1");
        child1.add(child1Data1);                    //添加到列表中
        Map<String,String> child1Data2 = new HashMap<String,String>();
        child1Data2.put("child", "child2");
        child1.add(child1Data2);                    //添加到列表中

        //定义一个 List，存放第二个一级标题需要的二级标题的数据
        List<Map<String, String>> child2 = new ArrayList<Map<String,
        String>>();
        Map<String, String> child2Data = new HashMap<String, String>();
        child2Data.put("child", "child1");
        child2.add(child2Data);

        //向二级标题添加数据
        childs.add(child1);
        childs.add(child2);
    }
}
```

最后，程序运行效果如图 5.30 所示。




初始化界面



展开后

图 5.30　可扩展列表的使用

5.3.5　实例——深入使用可扩展列表

通过上一小节的使用，读者对使用可扩展列表该是小有心得，可是现在的问题是：（1）我们仅仅展示了静态的数据，不能使用动态的数据添加。（2）虽然继承 ExpandableList Activity 可以提供一些方便，但很多时候也会带来很多不便，继承的越多，受限的也就越多。本小节将讲解的就是完全自定义的 ExpandableListView 的使用，从 Activiy 到 Aapter 都是自定义的，这样会带来很多自由和随性，读者可以细细体味。

本实例将实现的是类似于 QQ 好友列表的实现，首先编写 xml 代码。

friend.xml 代码：

该布局中仅仅只有一个可扩展列表。

```
<?xml version="1.0" encoding="utf-8"?>
<ExpandableListView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/expandable_list_view"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
>
</ExpandableListView>
```

接下来实现一级条目布局，Friend_group.xml 代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
  <TextView
    android:id="@+id/friend_group_name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ...//省略若干属性
    />

  <TextView
    android:id="@+id/friend_group_total"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ...//省略若干属性
    />
</LinearLayout>
```

其中 friend_group_name 文本框用来显示组名，friend_group_tota 文本框用来显示该组下的总人数，然后实现二级条目布局。

Friend_child.xml 代码：

二级条目只有一个 Textview，用来显示好友姓名。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
```

```
    >
  <TextView
    android:id="@+id/friend child name"
    android:layout width="wrap content"
    android:layout height="wrap content"
    ...          //省略部分属性
    />
</LinearLayout>
```

接下来的关键就是 Java 部分的代码了，首先完成整体设计。

1. 整体设计

```
package com.wes.expan;

import ...                                              //省略部分导入包
import java.util.*                                      //导入 java 工具包
import android.widget.ExpandableListView;
import android.widget.ExpandableListView.OnChildClickListener;

public class ExpanList extends Activity {
    private ArrayList<GroupData>    groupData;  //最后需要提供给适配器的数据
    private ArrayList<ChildData>    childData;  //二级标题的数据
    private List<String>            groupName;  //用于存放所有的一级标题
    public  ExpandableInfoAdapter   sela;               //适配器
    private final String            NAME  = "NAME";     //定义一些常量
    private final String            TOTAL = "TOTAL";
    private HashMap<String,String>  oriData;            //原始的未整理的数据
    public  ExpandableListView     mListView = null;    //可扩展列表对象

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.friend);
        loadData();                                     //调用加载数据函数
        initCtrl();                                     //调用初始化控件函数

        mListView.setOnChildClickListener(              //设置监听器
             new OnChildClickListener()
                      {
                          //此处完成功能设计
                      });
    }
```

整体设计简单明了。接下来，我们为了创建适配器简单些，实现了一些自己的内部类，用以保存数据：

```
     class ExpandableListData
    {                              //列表数据内部类，组织 adapter 需要使用的参数
        public List<GroupData> groupData;           //最后提供的数据源
        public int groupLayoutId;                   //一级标题的布局 ID
        public int childLayoutId;                   //二级标题的布局 ID
        public String[] groupFrom;                  //一级标题数据来源
        public String[] childFrom;                  //二级标题数据来源
        public int[] groupTo;                       //一级标题需要显示的组件 ID
```

```
        public int[] childTo;                          //二级标题需要显示的组件 ID

    public ExpandableListData(                         //该内部类的构造函数
                    List<GroupData> groupData,
                    int groupLayoutId,
                    String[] groupFrom,
                    int[] groupTo,
                 int childLayoutId,
                 String[] childFrom,
                 int[] childTo)
         {
         this.groupData = groupData;
         this.groupLayoutId = groupLayoutId;
         this.groupFrom = groupFrom;
         this.groupTo = groupTo;
         this.childLayoutId = childLayoutId;
         this.childFrom = childFrom;
         this.childTo = childTo;
         }
     }
```

一级标题数据内部类：

```
    class GroupData
    {                                                  //一级标题数据内部类
        public String NAME;
        public ArrayList<ChildData> CHILDDATA;
    }
```

二级标题数据内部类：

```
    class ChildData
    {                                                  //二级标题数据内部类
        public String NAME;
    }
```

通过内部类的组织，是不是对数据结构有了一个初步的了解了呢？暂时不理解没关系，我们接着往下看。

2. 完成初始化组件

这里完成了 4 个步骤：

（1）获得列表对象。

（2）新建了列表数据对象，也就是获得了新建的内部类的对象。

（3）新建适配器。

（4）设置适配器。

每个步骤的意义在上小节中已经讲解过，读者需仔细体会。

```
    protected void initCtrl()
    {
        mListView =(ExpandableListView) findViewById(R.id.expandable_
        list_view);                                            //实例化列表对象
        ExpandableListData ed = new ExpandableListData(        //新建列表数据对象
                groupData,                                      //数据源
                R.layout.friend_group,                     //一级标题布局文件
                new String[] {NAME, TOTAL},                //一级标题的 KEY
                new int[] { R.id.friend_group_name, R.id.friend_
```

```
            group total},                          //一级标题的组件 ID
            R.layout.friend_child,                 //二级标题布局文件
            new String[] {NAME},                   //二级标题的 KEY
            new int[] { R.id.friend_child_name});  //二级标题的组件 ID

        sela = new ExpandableInfoAdapter(this, ed);   //新建适配器
        mListView.setAdapter(sela);                   //设置适配器
    }
```

3. 完成数据加载

这依然是重头戏，我们必须把源数据转换成 adapter 可以解析的数据结构：

```
    protected void loadData()
    {
        oriData = new HashMap<String, String>();   //这里完成源数据的输入
        oriData.put( "刘静静","同学");
        oriData.put( "侍亮亮","同学");
        oriData.put( "杜其双","同学");
        oriData.put( "张三",  "同事");
        oriData.put( "李四",  "同事");
        oriData.put( "王五",  "同事");
        oriData.put( "胡淼",  "舍友");
        oriData.put( "赵强",  "舍友");
        oriData.put( "赵玉",  "女朋友");

        groupData = new ArrayList<GroupData>();    //最后需要提供的数据结构
        groupName = new ArrayList<String>();       //一级标题内容组成的列表
        ChildData cd = null;

        Iterator<Entry<String,String>> iter = oriData.entrySet().
        iterator();                                //序列化该 Map

        while(iter != null && iter.hasNext())      //遍历该 Map，组成列表
        {
            Entry<String,String> entry = iter.next();
            String group = entry.getValue();
            if(!groupName.contains(group))
            {
                groupName.add(group);              //若不重复则添加到列表中
            }
        }

        Map<String, ArrayList<ChildData>> map = new HashMap<String,
        ArrayList<ChildData>>();
                    //新建数据结构，一个一级标题，对应一组二级标题
        for(String str : groupName)                //遍历一级标题
        {
            childData = new ArrayList<ChildData>();
            map.put(str, childData);
                            //此时每个一级标题对应的二级标题列表中无数据
        }
        Iterator<Entry<String,String>> iter1 = oriData.entrySet().
        iterator();                                //重新序列化源数据
        while(iter1 != null && iter1.hasNext())    //遍历数据
        {
            cd = new ChildData();
```

```
            Entry<String,String> entry = iter1.next();
            String name = entry.getKey();          //得到二级标题内容
            String group = entry.getValue();       //得到一级标题内容
            cd.NAME = name;
            map.get(group).add(cd);     //找到对应的一级标题，添加到对应列表中
        }

        for(String str : map.keySet())             //根据 Key 遍历 Map
        {
            GroupData gd = new GroupData();
                                        //每个一级标题新建一个 GroupData 对象
            gd.NAME = str;
            gd.CHILDDATA = map.get(str);           //赋予对应的值
            groupData.add(gd);                    //将 GroupData 添加到列表中
        }
    }
}
```

笔者在本段代码中添加了详细的注释，基本上每句代码都有相应的注释，读者看一遍如果无法理解可以多多揣摩，书读百遍其义自现，这段代码虽然繁琐，但多看几遍应该可以完全理解。

4. 完成监听事件

形成了列表后，我们需要对每个选项进行监听，如 QQ。每个选项被单击后则打开聊天框，本例中就使用 Toast 显示被单击选项的名称了。

```
@Override
                        public boolean onChildClick(
                                ExpandableListView parent,
    //adapterView，可理解为父视图
                                View view,                 //被点击的视图
                                int groupPosition,         //一级标题的位置
                                int childPosition,         //二级标题的位置
                                long id)                   //该 View 的 id
                        {
                            String name = sela.getChildName(groupPosition,
                            childPosition);                //得到名字
                            Toast.makeText(getBaseContext(), "您即将与"+
                            name + "通话", Toast.LENGTH_LONG).show();
                            return true;
                        }
```

5. 自定义适配器实现

在 Spinner 的学习时，我们已经编写了一个自定义的 adapter，继承自 BaseAdapter，本例中实现的适配器需继承 BaseExpandableListAdapter。

需要重写若干方法，其中最主要的是两个方法：

（1）获得二级视图

```
public View getChildView(
            int groupPosition,              //一级标题位置
            int childPosition,              //二级标题位置
            boolean isLastChild,            //是否为最后一个二级标题
```

```
            View convertView,                    //需要转换的视图
            ViewGroup parent)                    //父视图
```

该方法通过 groupPosition、childPosition 得到二级标题的视图，是非常重要的一个方法。

（2）获得一级视图

```
@Override
    public View getGroupView(int position, boolean flag, View view,
    ViewGroup parent)
    {
}
```

该方法通过 position 获得一级标题的视图。

6. 完全代码

将所有的需要重写的 10 个方法重写后，就完成了自定义的 adapter。虽然看上去要做的工作量很大，但是部分的方法重写只要一句话就可以完成了，所以不要被打击了信心，勇敢地尝试一下吧！

```
package com.wes.expan;

import com.wes.expan.ExpanList.ChildData;                //导入 3 个内部类
import com.wes.expan.ExpanList.ExpandableListData;
import com.wes.expan.ExpanList.GroupData;

import ...                                               //省略部分导入包
import android.widget.BaseExpandableListAdapter;


public class ExpandableInfoAdapter extends BaseExpandableListAdapter
{
    private ExpandableListData ed;                       //数据源
    private Context    mContext  = null;   //上下文关系

    class ExpandableListChildView
    {
        TextView name;                    //一级视图
    }

    class ExpandableListGroupView
    {                                     //二级视图
        TextView name;                    //项目名称
        TextView total;                   //包含的子项目个数
    }

    public ExpandableInfoAdapter(Context context,ExpandableListData ed)
    {
        this.ed = ed;
        mContext  = context;
    }

    @Override
    public ChildData getChild(int groupPosition, int childPosition)
    {
        return ed.groupData.get(groupPosition).CHILDDATA.get
        (childPosition);
```

```
                            //根据一级位置和二级位置得到二级标题数据
    }

    @Override
    public long getChildId(int groupPosition, int childPosition)
    {                           //返回 childPosition
        return childPosition;
    }

    @Override
    public View getChildView(
            int groupPosition,              //一级标题位置
            int childPosition,              //二级标题位置
            boolean isLastChild,            //是否为最后一个二级标题
            View convertView,               //需要转换的视图
            ViewGroup parent)               //父视图
    {
        ExpandableListChildView elcv = null;
        if(convertView == null)
        {
            LayoutInflater inflater = LayoutInflater.from(mContext);
            convertView = (LinearLayout) inflater.inflate(ed.childLayoutId,
            null, false);                   //实例化需转换的视图
           elcv = new ExpandableListChildView();        //实例化二级标题视图
           elcv.name = (TextView) convertView.findViewById(ed.
           childTo[0]);                     //得到二级标题 name 的 View 对象
           convertView.setTag(elcv);                    //设置标签
        } else
          elcv = (ExpandableListChildView) convertView.getTag();

        ChildData cd = getChild(groupPosition, childPosition);
                                            //获得需显示的数据
        if(cd != null)
        {
            elcv.name.setText(cd.NAME);                 //显示数据
        }
        return convertView;                             //返回设置好的视图
    }

    @Override
    public int getChildrenCount(int groupPosition)
    {
        return ed.groupData.get(groupPosition).CHILDDATA.size();
                                                        //得到一级标题个数
    }

    @Override
    public GroupData getGroup(int groupPosition)
    {
        return ed.groupData.get(groupPosition);         //得到一级标题的数据
    }

    @Override
    public int getGroupCount()
    {
        return ed.groupData.size();                     //得到一级标题个数
    }
```

```
    @Override
    public long getGroupId(int groupPosition)
    {
        return groupPosition;                          //得到一级标题位置
    }

    @Override
    public View getGroupView(int position, boolean flag, View view,
    ViewGroup parent)
    {
        ExpandableListGroupView elgv = null;
        if (view == null) {
            LayoutInflater inflater = LayoutInflater.from(mContext);
            view =  inflater.inflate(ed.groupLayoutId, null, false);
                                                       //实例化一级标题布局
            elgv = new ExpandableListGroupView();
            elgv.name = (TextView)view.findViewById(ed.groupTo[0]);
                                                       //得到名称的视图
            elgv.total = (TextView)view.findViewById(ed.groupTo[1]);
                                                       //得到总计的视图
            view.setTag(elgv);
        }
        else
            elgv = (ExpandableListGroupView) view.getTag();
        GroupData gd = getGroup(position);             //得到对应的数据
        if (gd != null)
        {
            elgv.name.setText(gd.NAME);                //显示名称的内容
            elgv.total.setText("[ " + String.valueOf(getChildren
            Count(position)) + " ]");                  //显示总计的内容
        }
        return view;
    }


    @Override
    public boolean hasStableIds()
    {
        return true;                    //是不是所有相同的 id 总是指向同一个对象
    }

    @Override
    public boolean isChildSelectable(int groupPosition, int childPosition)
    {
        return true;                    //子选项是否可选
    }

    public String getChildName(int groupPosition, int childPosition)
    {
        return getChild(groupPosition, childPosition).NAME;
                                        //得到二级标题内容
    }

}
```

笔者对该段代码也进行了详细的注释，希望读者能耐心的阅读。最后需要提醒读者的是：如果 isChildSelectable()方法返回值为 false 的话，那么设置的 OnChildClickListener 方法是无效的，因为这个时候，子项不能被单击则无法捕捉单击事件。

总结发现：通过自定义的适配器可以实现很多功能，而 Simple 类的适配器则无法提供，所以虽然开头会很困难，但是不要气馁。当你融会贯通，你的编程能力又提高了一大步！

最后运行效果如图 5.31 所示。



初识界面　　展开后　　单击了子项后

图 5.31　好友界面

好了讲解到这里 ListView 的使用就告一段落了，接下来的工作就要各位读者自己去揣摩、去练习，不要害怕困难，当你克服了困难再回头看看，原来你眼里曾经的高山只是一个土丘。

5.4　使用菜单——Menu

学习了这么多的组件后，大家已经可以编写出一些非常实用的界面了，接下来我们将要讲解的是一个比较特殊的部分——Menu 的使用。

一般的 Android 手机都会有一个 Menu 按钮，我们即将展开的讲解就围绕着这个按钮展开。

5.4.1　Menu 的使用

平时在使用手机的时候，当按下 Menu 按钮后会弹出一个菜单。那菜单是怎么产生的呢？其实非常简单，步骤如下：

（1）重写 Activity.onCreateOptionsMenu(Menu menu)方法，产生其中的选项。在方法中调用 Menu.add(int groupId, int itemId, int order, CharSequence tittleRes)。

（2）添加菜单项。这里有 4 个参数：

- groupId：分组 Id，这里如果不需分组则填 0。
- itemId：选项的 Id，用来给监听器识别单击的是哪个菜单项。注意，该 Id 不可重复。
- order：排序信息，用来给菜单项排序，小的在前，大的在后。
- tittleRes：菜单项的显示信息。

（3）重写 Activity.onOptionsItemSelected(MenuItem item)方法，设置菜单项的监听事件。

通过 MenuItem.getItemId()方法可以获得被单击的菜单项的 Id，一起来区分即将进行的动作。

5.4.2　通过实例学习使用 Menu

本例中将模拟一个软件的 Menu 选项菜单，功能分别为跳转、隐藏视图和退出。需要注意的是，这里的跳转我们希望能分为“跳转到 A”和“跳转到 B”两个子选项。怎么实现呢？不要着急，马上来看讲解。

1. xml代码

在 main.xml 代码中添加一个文本控件，将其 id 设为 tv2。这一步非常简单，笔者就不再贴出代码，由读者自己去完成。

2. Java代码

在 Java 部分首先新建两个 Activity 用来跳转，分别将之命名为 Activity_A 和 Activity_B。接着在原 Activity 中完成整体设计：

```
package com.wes.menudemo;

import...                                    //省略部分导入包
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;

public class MenuDemo extends Activity {
    final int ITEM1 = 1;                     //菜单项的 id，注意不能重复
    final int ITEM2 = 2;
    final int ITEM3 = 3;
    final int ITEM4 = 4;
    final int ITEM5 = 5;
    TextView tv;                                             //文本框
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      tv = (TextView) findViewById(R.id.tv1);                //获得对象
         }

   public boolean onCreateOptionsMenu(Menu menu)
   {
    //此处添加菜单项
   }

   @Override
   public boolean onOptionsItemSelected(MenuItem item)
    {
    //此处添加单击事件监听
   }
}
```

接下来完成 Activity.onCreateOptionsMenu(Menu menu)方法的重写：

```
public boolean onCreateOptionsMenu(Menu menu)
    {
    super.onCreateOptionsMenu(menu);
    SubMenu subMenu = menu.addSubMenu(0, ITEM1, 0, "跳转");
    subMenu.add(2, ITEM4, 4, "跳转到 A");
    subMenu.add(2, ITEM5, 5, "跳转到 B");

    menu.add(0,ITEM2,1,"隐藏文本框");
    menu.add(0,ITEM3,2,"退出");
    return true;
    }
```

大家注意以下的一段代码：

```
SubMenu subMenu = menu.addSubMenu(0, ITEM1, 0, "跳转");   //添加二级菜单
subMenu.add(2, ITEM4, 4, "跳转到 A");                     //为二级菜单添加菜单项
    subMenu.add(2, ITEM5, 5, "跳转到 B");                 //为二级菜单添加菜单项
```

该方法即为子菜单的添加了。

接着我们完成菜单项单击事件的监听：

```
@Override
   public boolean onOptionsItemSelected(MenuItem item) {
    // TODO Auto-generated method stub
    switch (item.getItemId())
    {
    case ITEM1 :
        Toast.makeText(this, "sub Menu", Toast.LENGTH_SHORT).show();
        break;
    case ITEM2 :
        tv.setVisibility(View.INVISIBLE);              //将视图设为不可见
        break;
    case ITEM3 :
        finish();                                      //结束该活动
        break;
    case ITEM4 :
        Intent i = new Intent(MenuDemo.this,Activity_A.class);
        startActivity(i);                              //跳转到 A
        break;
    case ITEM5 :
        Intent intent = new Intent(MenuDemo.this,Activity_B.class);
        startActivity(intent);                         //跳转到 B
        break;
    }

    return true;
    }
```

如果是“跳转”菜单项被单击则告知用户，这是个 SubMenu，同时会弹出两个子菜单，分别显示“跳转到 A”和“跳转到 B”，如果被单击则完成跳转；

如果是“隐藏文本框”被单击，则将文本框设置为不可见；

如果是“退出”被单击，则结束该活动。

编写完毕！运行一下看看效果是不是和笔者的截图 5.32 一样呢。



出菜单项



单击跳转后弹出二级菜单

图 5.32　Menu 的使用

5.5 小　　结

本章首先讲解了视图、组件、视图容器等的概念和意义，以及它们之间的继承关系，使读者对 UI 编程有了一个初步的了解。接下来详细讲解了一些平常开发中经常需要使用的组件，比如文本框、编辑框、单选多选框，以及列表、可扩展列表等等。当然在实际开发中仅仅使用本书中介绍的一些组件肯定是不够的，更多的组件还需要读者自己去学习或者自己去开发。无论是查阅 Android SDK 或者上网搜索相关资料，在学习的过程中，相信读者的编程能力肯定会越来越强。

第 6 章　使用程序资源

资源是 Android 应用程序的一个重要组成部分，几乎每个程序都要使用资源。它包括字符串、图像、颜色、各类原始文件等，它还可以使代码更加容易阅读和管理，提高运行效率。在本章中，将会学到一系列与资源相关的知识，同时通过一些实例，掌握在 Android 应用程序中保存并使用资源的方法。

6.1　资源的意义

资源在 Android 编程中占有非常重要的地位。在 Android 中，资源与功能代码被分开，这是基于各方面的原因做出的决定：首先，它可以使程序便于管理和阅读；其次，部分资源会被编译到二进制代码中，提高了加载效率。

6.1.1　什么是资源

在日常生活中，我们随处可见“资源”这个词，煤矿资源、技术资源、人力资源等等。那么在 Android 中，资源又是指什么呢？它被用来指代那些在代码中被使用并被编译到应用程序中的外部文件。

可能这样说还有读者不是很理解，那么我们可以举一些简单的例子：在第 5 章的 ImageView 的学习中准备了一些图片与它关联，这些图片就是资源。再有，我们在编程中需要添加 xml 代码编写的布局文件，这些布局文件也是资源。当然，资源不止于这些，它还包括字符串以及一些原始数据，如音频文件、视频文件等。

6.1.2　怎样存储资源

大多数的资源都被存储于 xml 文件中，如字符串、颜色、尺寸、样式等。当然也可以使用原始数据文件来作为资源存储，如*.png、*.mp3 等。所有的资源文件都被保存在各自的子文件夹中，注意文件夹都必须以小写命名。而这些子文件夹都必须存储在/res 文件夹中，如图 6.1 所示。

如图 6.1 所示，所有的资源文件都被存在各自的子文件夹下，这些子文件夹又都在 res 文件夹下。

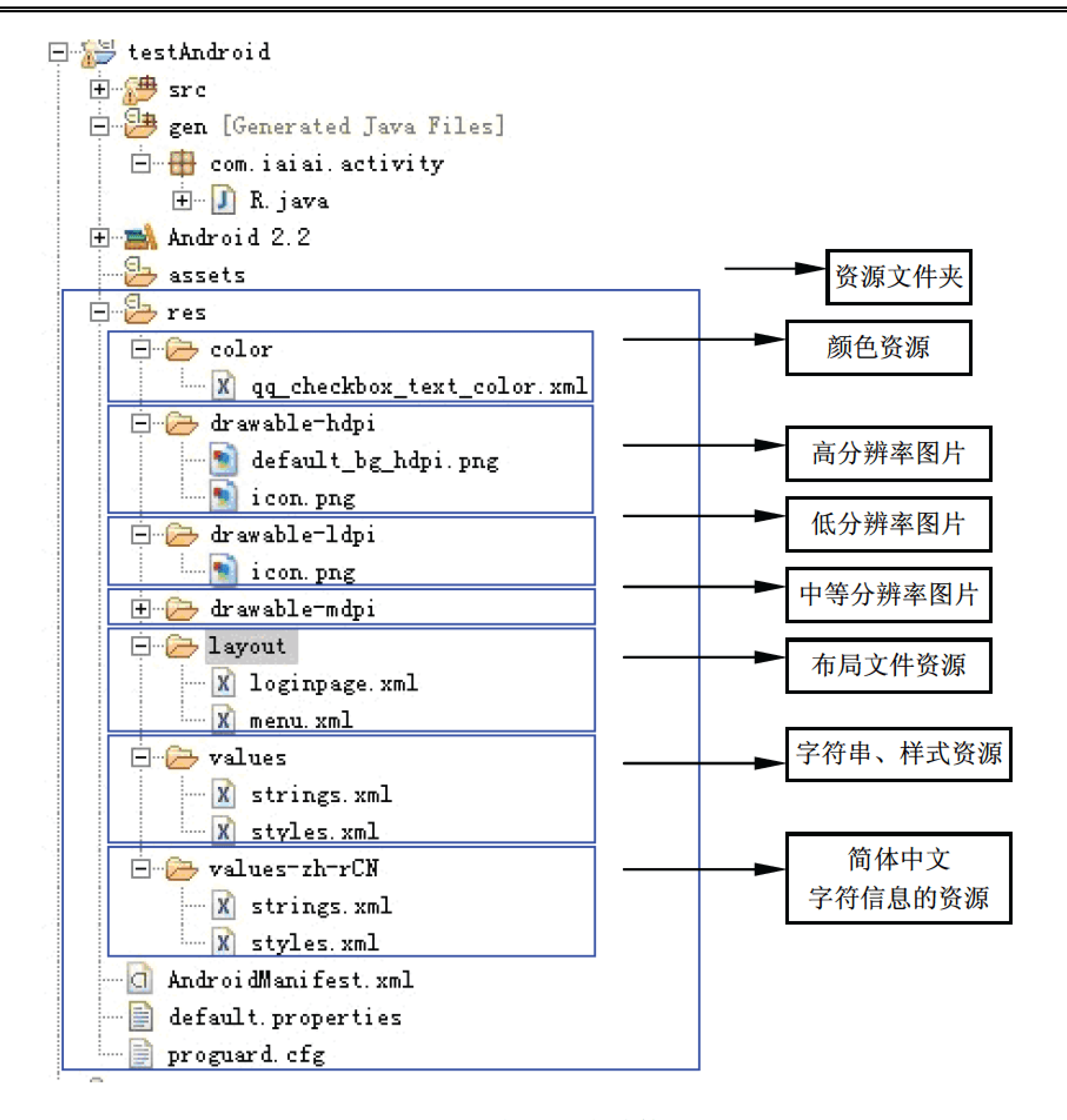


图 6.1　资源层次结构

6.1.3　怎样添加资源

本书的开发环境已经安装了 Android 开发工具插件——ADT（Android Development Tools Plug-In），所以添加资源非常的简单。当我们向工程资源目录/res 中添加新的资源时，插件会自动检测并通过 Android 组件打包工具——AAPT（Android Asset Packaging Tool）打包。更具体地说，是 AAPT 在后台编译了该资源，该资源提供了一个 Id 或者说是标示，使程序员在程序中可以顺利访问。

AAPT 可以将你的应用程序打包成 apk 文件以供安装，更可以自动将可识别的资源文件的 Id 编译成 R.java 文件。在图 6.1 中我们可以在/gen 目录下找到该文件，找到该文件并打开如下所示：

```
package com.iaiai.activity;

public final class R {
    public static final class attr {
    }
    public static final class color {
        public static final int qq_checkbox_text_color=0x7f060000;
    }
```

```
    public static final class drawable {
        public static final int bottom=0x7f020000;
        public static final int btn check off=0x7f020001;
        public static final int btn check off pressed=0x7f020002;
        public static final int btn check off selected=0x7f020003;
        public static final int btn check on=0x7f020004;
        public static final int btn check on pressed=0x7f020005;
        public static final int btn check on selected=0x7f020006;
        public static final int button2=0x7f020007;
        public static final int button2 down=0x7f020008;
        p
    }
    public static final class id {
        public static final int ImageButton02=0x7f070006;
        public static final int LinearLayout01=0x7f070001;
        public static final int RelativeLayout01=0x7f07000b;
        public static final int RelativeLayout02=0x7f070002;
        public static final int TextView01=0x7f070005;
        public static final int TextView02=0x7f070008;
        public static final int faceImg=0x7f070003;
        public static final int loginRoot=0x7f070000;
        public static final int login btn login=0x7f07000a;
        public static final int login cb savepwd=0x7f070009;

    }
    public static final class layout {
        public static final int loginpage=0x7f030000;
        public static final int menu=0x7f030001;
    }
    public static final class string {
        public static final int app name=0x7f040001;
        public static final int hello=0x7f040000;
        public static final int login=0x7f040006;
        public static final int opt_acceptGroupMsg=0x7f04000a;
        public static final int opt hideLogin=0x7f040007;
        public static final int opt_openVibra=0x7f040009;

    }
    public static final class style {
        public static final int MyCheckBox=0x7f050000;
    }
}
```

这是一段任意的 R 文件，我们可以发现其中都是一些常量。而这些常量就是该资源的 ID，在程序中我们需要依赖这些 ID 才能找到相应的资源。例如，在使用 TextView01 时：我们使用 findViewById(R.id.TextView01)来找到该资源；在使用 loginpage 时，我们使用 findViewById(R.layout.loginpage)等等。这里的 R 就是指该 R 文件，而 id、layout 都是该 R 文件中的相关的类。与图 6.1 对应，每个子文件夹都能在 R 文件中找到对应的类。值得注意的是：这些常量都不能被修改，在资源发生改变时，它会自动改变。

如果没有安装 ADT，那么很不幸，你还需要在命令行模式下对资源文件进行编译以便程序使用。所以强烈建议大家安装 Android 开发工具插件。

6.1.4　资源的种类

Android 中的资源有很多用途，如提供给用户的界面设计、图像显示、配色方案等等。

大部分情况下，我们都使用 xml 文件作为资源被使用。对于各自的子文件夹我们有一些约定俗成的规则，比如类名最好大写、方法名最好小写一样，每个资源文件名称必须是小写并尽量地简单易懂。何谓简单，简单就是仅使用字母、数字以及下划线组成文件名，易懂就是尽量使用英文缩写或全拼，如颜色，可以将文件夹命名为 color，资源文件命名为 color.xml，如表 6-1 所示。

表 6-1　一些常用的资源文件名称及类型

资 源 类 型	文件夹名称	文　件　名
字符串	/res/values/	strings.xml
字符串数组	/res/values/	arrays.xml
颜色	/res/values/	colors.xml
样式	/res/values/	styles.xml
主题	/res/values/	themes.xml
图片	/res/drawable/	*.png、*.jpg、*.gif 等
布局	/res/layout/	如 mylayout.xml 等
动画	/res/anim/	如*_anim.xml
菜单	/res/menu/	如 menu1.xml 等
原始文件	/res/raw/	如 audio.mp3、video.mp4
xml 文件	/res/xml/	自定义的 xml 文件

相信通过表 6-1，读者对一些经常使用的资源类型已经有了一个初步的了解，接下来我们将讲解怎样在程序中访问这些资源。

6.1.5　怎样访问资源

前文我们说过，在 Android 开发环境中，AAPT 会帮助我们将资源在后台编译并生成 R 文件，我们可以通过 R 文件中的 Id 来顺利地找到该资源。例如，一个 helloworld 的字符串资源可以在代码中作如下引用：

```
R.string.helloworld
```

如需得到该引用具体的值需要以下步骤：

（1）获得资源实例，方法为：

```
ContextWrapper.getResources()
```

通过该方法即可得到 Resources()实例，而所有的 Activity 都继承自 Context 类，所以可以直接使用 getResources()方法。

（2）通过对应方法得到相应类型的资源。

例如，要得到 String 类型，可以使用方法：

```
Resources.getString(int id)
```

这里的参数 id 就是 R.string.helloworld 了。

（3）再例如得到颜色类型，可以使用方法：

```
Resources.getColor(int id)
```

同样的，这里的 id 应该是 R.color.*。

（4）综上所述，如果要得到 helloworld 字符串资源中的数据，方法为：

```
String myString = getResources().getString(R.string.helloworld)
```

关于资源的基础知识我们就讲解到这里，接下来我们将学习的是如何合理而高效地在程序中使用资源。

6.2 使 用 资 源

本节中，读者将学习如何使用各类资源，下面将详细讲解各类资源类型的定义和使用。通过合理的使用资源，使用者才可以编写出更加丰富的界面。而编写资源的两种方法——使用资源编辑器和直接编写 xml 文件，在本节中也会有详细的介绍。

6.2.1　使用资源管理器

我们知道使用资源可以有两种方法，第一种是使用 Android 插件——资源管理器。使用资源管理器的好处是省去了大部分的代码编写工作，缩短了开发时间，减少了开发人员的工作量。缺点是使用它只能编写出简单的属性，不能随意定制。

接下来将学习的是通过资源管理器快捷地创建资源。首先我们认识一下资源管理器的庐山真面目：

打开任意一个工程文件，在/res/values 文件夹中会保存有一个 strings.xml 文件。什么？你没有创建它？没有关系，这个文件是系统默认创建的，它会在里面保存该应用的名称字符串以及默认的 hello 字符串。

打开它的同时你已经打开了 Android 提供的资源管理器插件，如图 6.2 所示。

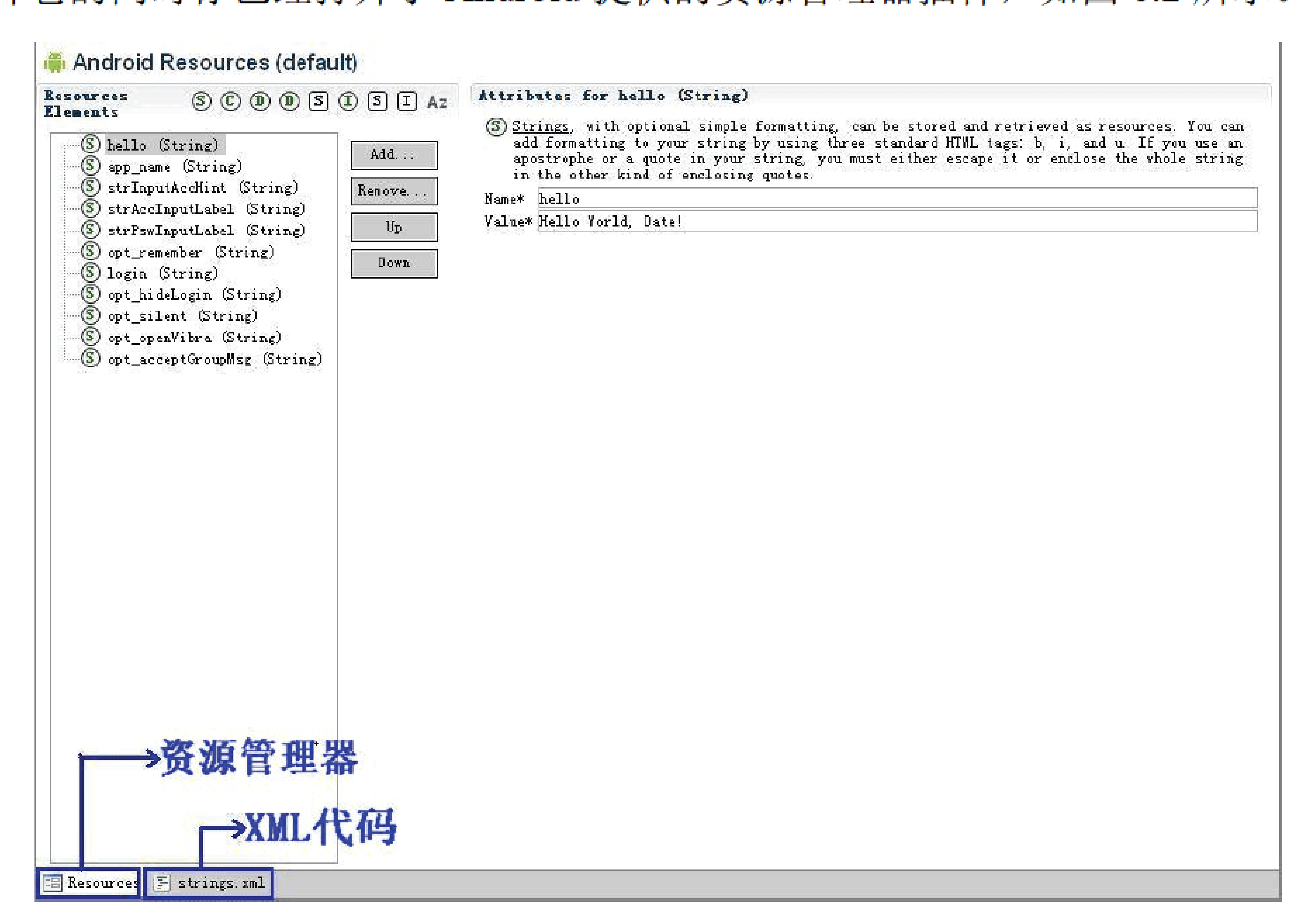


图 6.2　Android 资源管理器

使用它非常简单，左侧显示了该 xml 文件中包含有的资源，右侧显示了选中的资源的属性，以及一些注意事项，属性中包括 name 属性以及 value 属性。

使用资源管理器固然是方便，但是我们也不能过分依赖它，因为资源管理器也不是万能的，它可以创建的资源种类是有限的。单击 Add 按钮选择添加资源，显示如图 6.3 所示。

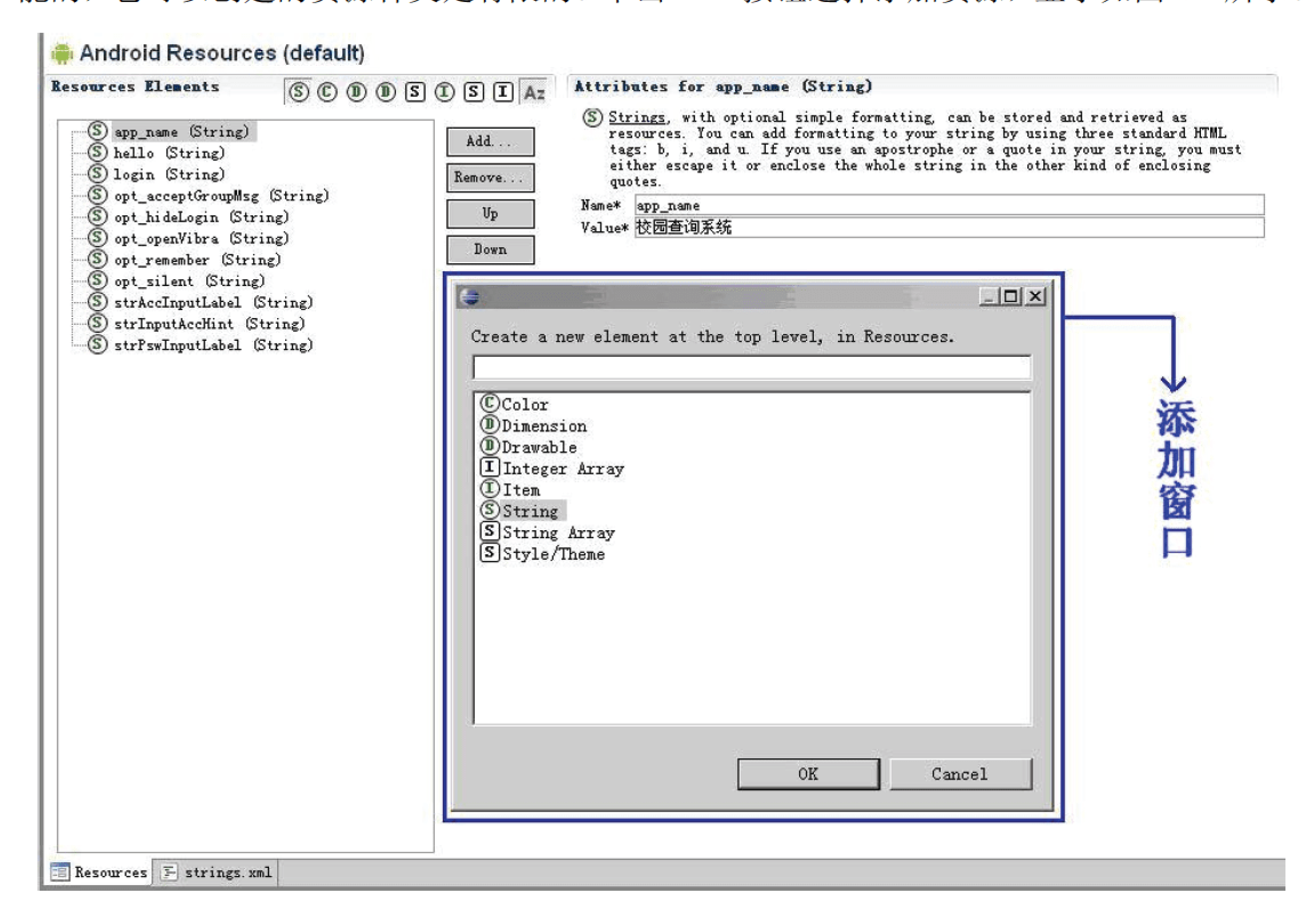


图 6.3　单击 Add 按钮后

我们可以发现，Android 资源管理器可以创建的资源种类包括 8 类，如表 6-2 所示。

表 6-2　Android资源管理器的种类

类　　型	意　　义
Color	颜色资源
Dimension	尺寸资源
Drawable	图片资源
Integer Array	整数数组资源
Item	选项资源
String	字符串资源
String Array	字符串数组资源
Style/Theme	样式/主题资源

除了表 6-2 列出的 8 种类型，Android 资源管理器就无能为力了。选择创建的时候，只需在弹出框中选择你希望创建的类型，然后在右侧的编辑框中输入属性，系统就会为你自动编写 xml 代码，节省了开发人员大量的精力。

具体的创建流程我们以 String 型为例：

（1）在/res/values 文件夹下创建一个 xml 文件，命名规则我们在上一节中已经有了

讲解。

（2）通过打开 xml 文件打开 Android 资源管理器。

（3）单击 Add 按钮。

（4）在弹出框中选择你希望创建的类型。

（5）在右侧的属性栏中输入 name 和 value 属性。

（6）保存设置，创建就完成了。

各个类型的资源创建大同小异，由于本书篇幅有限，这里就不再一一讲解了，读者可以自行学习。

6.2.2　使用 String 资源

我们依旧从最简单的 String 资源开始资源学习之旅。String 资源，顾名思义，就是用来显示一些字符串，常被用于帮助信息和提示信息或文本标签等。我们每次新建工程时，系统会同时新建一个 String 资源，用作存储应用程序名。

要使用 String 资源需要经过如下两个步骤：

1. 定义资源

（1）在/res/values 工程目录下定义一个 strings.xml 文件夹，在其中完成 string 资源的定义。语法如下：

```
<resources>
    <string name="*** "> value</string>
</resources>
```

记住所有的资源文件都必须在<resources>节点下。string 资源则使用<string>标签，注意这里的 string 首字母是“s”而不是大写的“S”。这与 Java 代码不同，在 Java 代码中 string 必须大写因为它是一个类，而在 xml 代码中 string 只是一种资源类型而非一个类。

在<string>标签中需要有两个属性：

- name 属性：也就是该属性的 Id，R 文件通过该 name 才能正确找到实际的内容。
- value 属性：也就是在<string>和</string>标签中的内容。

这里需要提醒广大读者朋友们的是：在添加 value 属性时可以直接输入你希望显示的字符而不需要使用双引号，但在这里又建议大家使用双引号，为什么呢？

原因有二：一者是从养成的编程习惯的角度来考虑，二者是从转义的角度来考虑。那何谓转义呢？因为 String 资源在编译时会被编译到应用程序中，所以当你希望显示包含单引号或撇号的字符串时需要进行转义。有时候开发人员没有注意这一点会引起不必要的错误，所以为了简化开发，建议大家直接用双引号将需要显示的内容包含，这样就不再有后顾之忧了。

当然，如果读者朋友不喜欢使用双引号，那也可以采用麻烦一些的转义。例如，要显示如下内容：

```
Please enter user’s password:
```

我们需要使用如下方法定义 string 资源：

```
<string name="*** "> Please enter user\'s password : </string>
```

再举一个例子，我们要显示如下内容：

```
注意：‘’以及“”不能被使用！
```

我们需要使用如下方法定义 string 资源：

```
<string name="*** ">注意：\‘\’以及\“\”不能被使用！</string>
```

简单的资源定义方法介绍到这里就结束了，接下来要介绍的是另外的三种属性——粗体、斜体和下划线。这 3 个属性是 html 中的风格，分别用节点<b>、<i>和<u>来标示。

例如：

```
<string name="text "> <b>粗体 Bold</b>, <i>斜体 Italia</i>, <u>下划线 Underline</u></string>
```

当然你也可以同时使用这 3 个属性，例如：

```
<string name="text2"><i><u><b>粗体，斜体，下划线</b></u></i></string>
```

同时使用的时候请注意各个节点的配对问题，否则属性将无效。

2．引用资源

定义好资源文件之后我们就要使用它们，那么如何使用它们呢？有两种方法：

（1）在 xml 代码中使用

我们以 TextView 为例，语法格式如下：

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/text"
    />
```

这里的“@”前缀声明这是一个资源引用。也就是说，随后的文本是以@[package:]type/name 形式提供的资源名，而非具体要显示的内容。在实际的编程中我们经常不需要指明特定的包，因为我们通常只是在自己的包中引用。在 xml 代码中引用 string 资源就是这么简单，接下来我们学习在 Java 代码中使用 string 资源。

（2）在 Java 代码中使用 string 资源

在上一节中已经提到，在 Java 中，使用 ResourceManger 来管理资源，通过：

```
getResources().getString(R.sring.text)
```

就可以得到事先定义好的 string 资源。

6.2.3　实例——彩虹和太极

学习完理论知识以后，我们照例继续通过一个实例来完成对知识的巩固。在本例中将通过资源调用的方法来显示出彩虹的七种颜色和太极的两种颜色。

首先是资源文件 strings.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app name">ResDemo</string>
<string name="rainbow"><b>彩虹有七种颜色：</b></string>
   <string name="red">赤色</string>
   <string name="orange">橙色</string>
   <string name="yellow">黄色</string>
   ......                         //此处省略了其他颜色的定义
   <string name="taichi"><i><u><b>太极有两种颜色：</b></u></i></string>
   <string name="white"><u>白色</u></string>
   <string name="black"><u>黑色</u></string>
</resources>
```

无论是使用 Android 提供的资源管理器或者是直接在 xml 代码中复制粘贴，完成这一步都会非常的简单。注意这里 name 为 taichi 的资源，我们同时使用了粗体、斜体和下划线三种风格。

接下来是布局文件 main.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent"
   >
<TextView
   android:layout width="fill parent"
   android:layout height="wrap content"
   android:text="@string/rainbow"
   />
<TextView android:text="@string/red"
android:id="@+id/textView1"
android:layout width="wrap content"
android:layout height="wrap content"></TextView>
<TextView
android:text="@string/orange"
android:id="@+id/textView2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></TextView>
......                        //此处省略部分组件
<TextView
android:text="@string/taichi"
android:id="@+id/textView8"
android:layout width="wrap content"

android:layout height="wrap content"></TextView>
......                        //此处省略部分组件
</LinearLayout>
```

注意其中的粗体部分即为资源的引用，这样做条理非常清晰，程序结构一目了然，为以后的程序阅读提供了很多便利。最后是 Java 代码，实际上 Java 代码中不需再做其余的工作了，这里就不再贴出代码了。运行效果如图 6.4 所示。

OK，程序运行非常成功，但是这么做还是显得有些繁琐，重复的劳动比较多，效率还是显得比较低，接下来我们将使用 String 数组来简化开发。



图 6.4　彩虹和太极的颜色

6.2.4　使用 String 数组资源

String 数组也就是一串字符串列表，经常被使用于 menu 和 spinner 选项的保存。在上小节的例子中，如果使用 String 数组会取得一个比较理想的效果。要使用 String 数组资源同样要两个步骤：

（1）定义 string-array 资源。String 数组资源是用的标记为<string-array>，其下又包含有若干个选项，也就是<item>标记，每一个 item 包含有一个字符串。语法格式如下：

```
<string-array name="name">
    <item>item1 </item>
    <item>item2</item>
    <item>item3</item>
  </string-array>
```

（2）在代码中引用资源。同样的，我们使用：

```
getResource().getStringArray(R.array.name)
```

得到一个 String 数组。

接下来我们就来改进上小节中的例子。首先我们新建一个 arrays.xml 文件，用于保存 String 数组，其中添加代码如下：

（1）arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="rainbowcolors">
    <item>赤色</item>
    <item>橙色</item>
    <item>黄色</item>
  ......                            //省略部分 item
  </string-array>
  <string-array name="taichicolors">
    <item>白色</item>
    <item>黑色</item>
  </string-array>
</resources>
```

在每个 item 中我们已经把需要定义的字符串都定义好了，这样就省去了重复的资源引用工作，节约了开发时间。接着是 main.xml 代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
     android:id="@+id/textView0"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/rainbow"
    />
<TextView
```

```
    android:text="@string/taichi"
    android:id="@+id/textView1"
    android:layout width="wrap content"
    android:layout height="wrap content">
</TextView>
</LinearLayout>
```

没有错，就只有两个 TextView，并没有省略其他的文本框，两个就够了。接着在 Java 代码进行操作。

（2）Java 代码

第一步是得到定义的 String 数组：

```
String[] rainarray = getResources().getStringArray(R.array.
rainbowcolors);
String[] array = getResources().getStringArray(R.array.taichicolors);
```

然后获得 TextView 的操作对象：

```
TextView tv0 = (TextView) findViewById(R.id.textView0);
TextView tv1 = (TextView) findViewById(R.id.textView1);
```

最后通过遍历将数组显示到 TextView 中：

```
for(String s:rainarray)
{
   tv0.append("\n"+s);
}
for(String s:array)
{
   tv1.append("\n"+s);
}
```

好了，完成了，让那些多余的 TextView 都消失吧，我们只需要两个就完成了同样的效果！运行一下，看看效果是不是一样呢？如果程序正常运行，效果应该和图 6.4 一样，就不再贴出效果图了。

接下来我们将为各个颜色真正“着色”，让效果看起来更加丰富和逼真。

6.2.5　使用 Color 资源

Android 经常使用颜色资源来使界面更加华丽，那么这些颜色资源在 xml 文件中是怎样被存储的呢？事实上，颜色的值是一组按一定格式保存的十六进制数。那么所谓的格式又是什么呢？在 Android 中，我们规定颜色值按照 RGB 格式表示。

RGB 是一种颜色的表示模式，在工业界被广泛使用。我们都知道三原色是红、绿、蓝，通过不同的颜色搭配可以组成各种各样的颜色。而这，也就是 RGB 模式的基本原理了。所以 RGB 也就是 Red、Green、Blue 的缩写了。

当然，在 Android 中我们还可以使用 Alpha 值来表示透明度，0 最小，表示完全透明。基本知识介绍完后，我们就可以编写具体的颜色值了，Android 支持 4 种表示方式，它们都以#开头：

- ❑ #RGB（如#000，表示 12bit 的白色）
- ❑ #ARGB（如#8F00，表示 12bit 的红色，透明度为 50%）
- ❑ #RRGGBB（如#00FF00，表示 24bit 的绿色）

- #AARRGGBB（如#800000FF，表示 24bit 的蓝色，透明度为 50%）

1．在xml文件中定义颜色

OK，接下来我们学习怎样在 xml 文件中定义颜色，其语法如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="black">#FF000000</color>
   <color name="white">#FFFFFFFF</color>
</resources>
```

定义颜色时，同样采用了键值对的形式。在<color>标记下，将前文讲解的颜色格式填入就可以了。使用方法同样有两种：

（1）在 xml 代码中的方法为：

```
android:textColor="@color/black"
```

（2）在 Java 代码中的方法为：

```
int white = getResources().getColor(R.color.white);
```

注意颜色是 int 型的，而不是 Color 型。事实上，的确存在 Color 类，该类中保存有一些经常使用的颜色常量，其实这些常量同样是 int 型。

2．示例的改进

接下来我们就可以着手进行示例的改进了。首先新建一个 colors.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="black">#FF000000</color>              //黑色
   <color name="red">#FFFF0000</color>                //红色
   <color name="green">#FF00FF00</color>              //绿色
   <color name="blue">#FF0000FF</color>               //蓝色
   <color name="yellow">#FFFFFF00</color>             //黄色
   <color name="pink">#FFFF00FF</color>               //粉红色
   <color name="cyan">#FF00FFFF</color>               //青色
   <color name="purple">#FF800080</color>             //紫色
   <color name="orange">#FFFFA500</color>             //橙色
   <color name="gray">#FF808080</color>               //灰色
   <color name="white">#FFFFFFFF</color>              //白色
</resources>
```

这里只是列出了一些本例中需要使用的颜色，其他更多的颜色大家可以上网查找其具体的 RGB 值。接下来在 main.xml 代码中添加颜色的引用：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="@color/gray"
   >
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
```

```
    android:text="@string/rainbow"
    />
<TextView
android:textColor="@color/red"
android:text="@string/red"
android:id="@+id/textView1"
android:layout width="wrap content"
android:layout height="wrap content"></TextView>

......                          //此处省略部分组件

<TextView android:text="@string/taichi"
android:id="@+id/textView8"
android:layout_width="wrap_content"
android:layout height="wrap content">
</TextView>

<TextView
android:text="@string/white"
android:id="@+id/textView9"
android:layout width="wrap content"
android:layout height="wrap content"></TextView>
<TextView
android:text="@string/black"
android:id="@+id/textView10"
android:layout width="wrap content"
android:layout height="wrap content"></TextView>
</LinearLayout>
```

注意这里的 textView1～textView7 都使用了 android:textColor="@color/**"方法得到颜色的引用，而 textView9 和 textView10 则没有使用颜色引用，这是为了在代码中进行着色。这样可以同时巩固学习两种方法。

3. Java代码中着色

在 Java 代码中着色需要 3 个步骤：

首先得到 TextView 的操作对象：

```
TextView tv9 = (TextView) findViewById(R.id.textView9);
       TextView tv10 = (TextView) findViewById(R.id.textView10);
```

接着得到颜色的引用：

```
       int white = getResources().getColor(R.color.white);
       int black = getResources().getColor(R.color.black);
```

最后为文字着色：

```
       tv9.setTextColor(white);
       tv10.setTextColor(black);
```

好啦，到这里两种方法我们都使用过了，赶快运行一下看看效果如何吧！效果应该如图 6.5 所示。

6.2.6　使用 Dimension 资源

在应用程序中我们不可避免地要与一些尺寸大小打交道，那么在 Android 中尺寸又是

以什么为单位呢？我们又是如何定义和使用尺寸资源的呢？这就是本小节要讲解的内容。



图 6.5　使用颜色引用

首先，我们从表 6-3 中大略地了解 Android 中的一些尺寸单位。

表 6-3　Android支持的尺寸单位

单　　位	标　　记	说　　明
毫米	mm（milli meters）	实际的物理测量单位，在屏幕上以毫米为单位显示
英寸	in（inch）	实际的物理测量单位，在屏幕上以英寸为单位显示，1 英寸约为 25.4 毫米
像素	px（pixels）	不同的设备显示效果相同，是 Android 屏幕显示的默认单位，一般手机分辨率为 320×480 像素或者是 480×720 像素
点	pt（point）	标准的长度单位，1 点等于 1/72 英寸，常用于印刷业
密度独立像素	dp（density independed pixels）	实际上这个单位也是像素，只不过与设备有关了，也就是说每个设备都有其独立的像素
刻度独立像素	sp（scale independed pixels）	这类像素与密度无关，与刻度也无关，常被用于字体的显示，Android 字体大小默认单位为 sp

我们假设屏幕密度为 160dpi（dot per inch），也就是每英寸有 160 个点。这个时候 1sp=1dp=1px，事实上，Android 中 sp、dp 都是以 160dpi 为标准设定的。假设屏幕密度为 320dpi，这个时候，1dp = 1sp = 1×(320/160)px = 2px。其中 320/160 是密度比例因子。

在 6.1.2 小节中，我们已经知道，在/res 文件夹下包含有 3 个 drawable 文件夹，分别是：

- drawable-hdpi：高分辨率图片资源。
- drawable-mdpi：中分辨率图片资源。
- drawable-ldpi：低分辨率图片资源。

那么这些资源分别在什么时候被使用呢？事实上：

- 当屏幕密度为 240 时，系统使用 hdpi 标签中的资源。
- 当屏幕密度为 160 时，系统使用 mdpi 标签中的资源。
- 当屏幕密度为 120 时，系统使用 ldpi 标签中的资源。

1．定义尺寸资源

接下来学习定义尺寸资源，其语法格式如下：

```
<dimen name="pt">20pt</dimen>
```

与之前学习的 String 或 Color 资源类似，Dimension 资源在定义时同样以键值对的形式保存，不同的是 Dimension 的值需要附带单位。具体选择怎样的单位就需要开发者自行斟酌处理，可参考表 6-3。

在代码中使用尺寸资源同样非常简单，在 xml 代码中通过：

```
android:textSize="@dimen/pt"
```

将字体大小设置为 name=“pt”的尺寸，结合之前定义的内容，该语句实际作用是将字体大小设置为 20pt。

在 Java 代码中通过：

```
float  myDimen = getResources().getDimension(R.dimen.pt);
```

得到名为 pt 的尺寸资源。

2．加工且验证尺寸资源

同样地，我们继续对之前的实例进行加工以验证尺寸资源的使用效果。首先创建 dimens.xml 文件，在其中添加如下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <dimen name="pt">20pt</dimen>
   <dimen name="dp">20dp</dimen>
   <dimen name="sp">20sp</dimen>
   <dimen name="px">20px</dimen>
   <dimen name="mm">10mm</dimen>
   <dimen name="in">0.5in</dimen>
</resources>
```

注意这里设置的值可以是小数，假如设置为 1in 可能屏幕就无法显示完整内容了。

接下来是修改 main.xml 中的代码，限于篇幅，只列出各组件的 id 以及 textSize 属性，其代码片段如下：

```
<TextView    android:textSize="@dimen/pt"    android:id="@+id/textView2"
></TextView>
<TextView    android:textSize="@dimen/dp"    android:id="@+id/textView3"
></TextView>
<TextView    android:textSize="@dimen/sp"    android:id="@+id/textView4"
></TextView>
<TextView    android:textSize="@dimen/px"    android:id="@+id/textView5"
></TextView>
<TextView    android:textSize="@dimen/mm"    android:id="@+id/textView6"
```

```
></TextView>
<TextView   android:textSize="@dimen/in"     android:id="@+id/textView7"
></TextView>
```

好了，Java 部分就不做修改了，如果有读者朋友希望通过 Java 代码修改字体大小，不妨通过之前讲解的方法试验一下，看看能不能达到自己的目的吧！

最后运行结果如图 6.6 所示。

通过观察图片，我们可以得到一个推论：这里所用的模拟机其屏幕密度是 160，因为 px、dp、sp 大小相等。



图 6.6　使用尺寸资源

6.2.7　使用 Drawable 资源

Drawable 资源是 Android 资源中一个很大的课题，它包括很多方面，在这里就将其大体分为两个部分：

（1）简单的 Drawable 资源，通过 xml 代码编写，其效果与 color 类似。

（2）图片资源，这里的图片可以是 png、9.png、jpg、gif 等格式。

首先，我们先讲解简单的 Drawable 资源，如前所述，该类 Drawable 资源保存在/res/values 文件夹下。

1．简单Drawable资源

其语法格式与 Color 资源的定义类似：

```
<drawable name="gray_rect">#FF808080</drawable>
```

这样就定义了一个灰色的矩形框，这个时候我们可以使用：

```
android:background="@drawable/gray_rect"
```

将屏幕背景设置为一个灰色矩形，这点与 color 效果非常类似。你同样可以使用：

```
ColorDrawable   myDraw   =   (ColorDrawable)   getResources().getDrawable
(R.drawable.gray_rect);
```

在 Java 代码中得到该资源的引用并使用它。

2．图片资源

应用程序中我们经常需要使用一些图片或者图标来丰富我们的界面，在 Android 中一共支持 4 种图片格式，它们如表 6-4 所示。

表 6-4　Android支持的图片格式

格　　式	扩　展　名	说　　明
便携式网络图像	png（Portable Network Graphics）	无损的图片格式（推荐）
9 格拉伸图像	9.png（Nine-Patch stretchable Images）	由 png 格式转换而来，无损（推荐）

续表

格　　式	扩　展　名	说　　明
联合图像专家组	jpg（Joint PhotoGraphic Experts Group）	有损的图片格式，不推荐但可以接受
图形交换格式	gif（Graphics Interchange Format）	基本不被使用的图片格式

这些图片都被保存在/res/drawable 文件夹下，更确切地说，在屏幕密度为 160 时保存在/res/drawable-mdpi 文件夹下，屏幕密度为 240 时保存在/res/drawable-Hdpi 文件夹下，屏幕密度为 120 时保存在/res/drawable-ldpi 文件夹下。这在前文已经提及，如果仍有疑问请翻阅上小节。

图像资源在很多时候被称为 BitmapDrawable，也就是位图资源。添加图像文件非常简单，只需将需要的文件复制粘贴到前文所讲的文件夹下就可以了。使用它们同样非常简单，我们只需将文件名作为 Id 就可以了。在 xml 代码中，其语法格式与简单 Drawable 资源的使用殊无二致。在 Java 代码中稍有不同，事实上在上一章学习 ImageView 时，我们已经使用了相关的代码：

```
ImageView myView = (ImageView) findViewById(R.id.ImageView1);
myView.setImageResource(R.drawable.icon);
```

这样我们就已经将图片名为 icon 的图片显示在 ImageView 上了。

当然，我们也可以使用：

```
BitmapDrawable bitmapDraw = (BitmapDrawable) getResources().getDrawable
(R.drawable.icon);
```

得到名称为 icon 的图片资源，并在之后的代码中使用它。比如得到图片的高度：

```
int height = bitmapDraw.getIntrinsicHeight();
```

好了，学习完之后我们赶紧来试一试吧！我们依旧使用之前的实例，还记得学习 String 数组时的代码吗？在 main.xml 文件中我们放了两个 TextView，然后使用了 String 数组资源很简洁地完成了我们需要的效果。

接下来，我们为两个 TextView 添加背景图片，这样效果看起来更加丰富。首先准备两张图片分别为彩虹和太极，如图 6.7 所示。

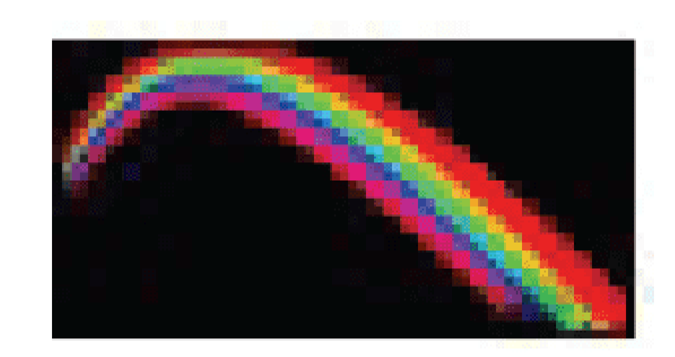



图 6.7　彩虹和太极

接着我们修改 main.xml 中的代码，添加如下代码：

```
<TextView
     android:id="@+id/textView0"
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:text="@string/rainbow"
    android:background="@drawable/rainbow1"
    />
```

```
<TextView
    android:text="@string/taichi"
    android:id="@+id/textView1"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:background="@drawable/taichi">
</TextView>
```

OK，Java 部分我们不需再做什么修改，运行一下，看看效果是不是如图 6.8 所示呢。

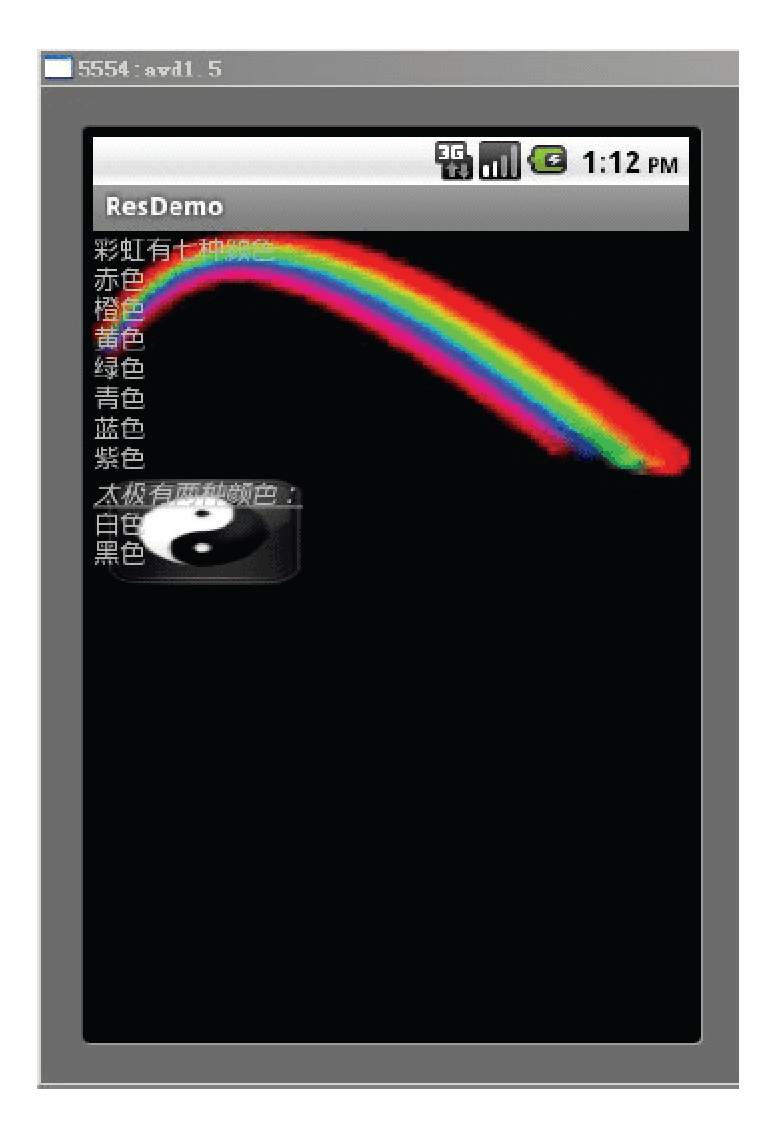


图 6.8　使用 drawable 资源

我们仔细观察太极图案会发现，这个太极被拉伸得有一些变形，所以图片看上去有一些别扭，而这样的效果我们肯定是不满意的，那怎么办呢？这个时候我们就要使用 9.png 格式的图片来应对这种情况。因为使用 9.png 格式的图片可以指定我们希望被拉伸的地方，而不希望被拉伸的地方则保持不变，这样图案就不会失真了。

要制作 9.png 格式的图片需要使用 Android SDK 中提供的一个工具，名为 draw9patch，位于 SDK 的/tools 文件夹内。接下来笔者将讲解该工具的使用：

（1）运行 Android SDK Tools 目录下的 draw9patch.bat 文件，运行效果如图 6.9 所示。

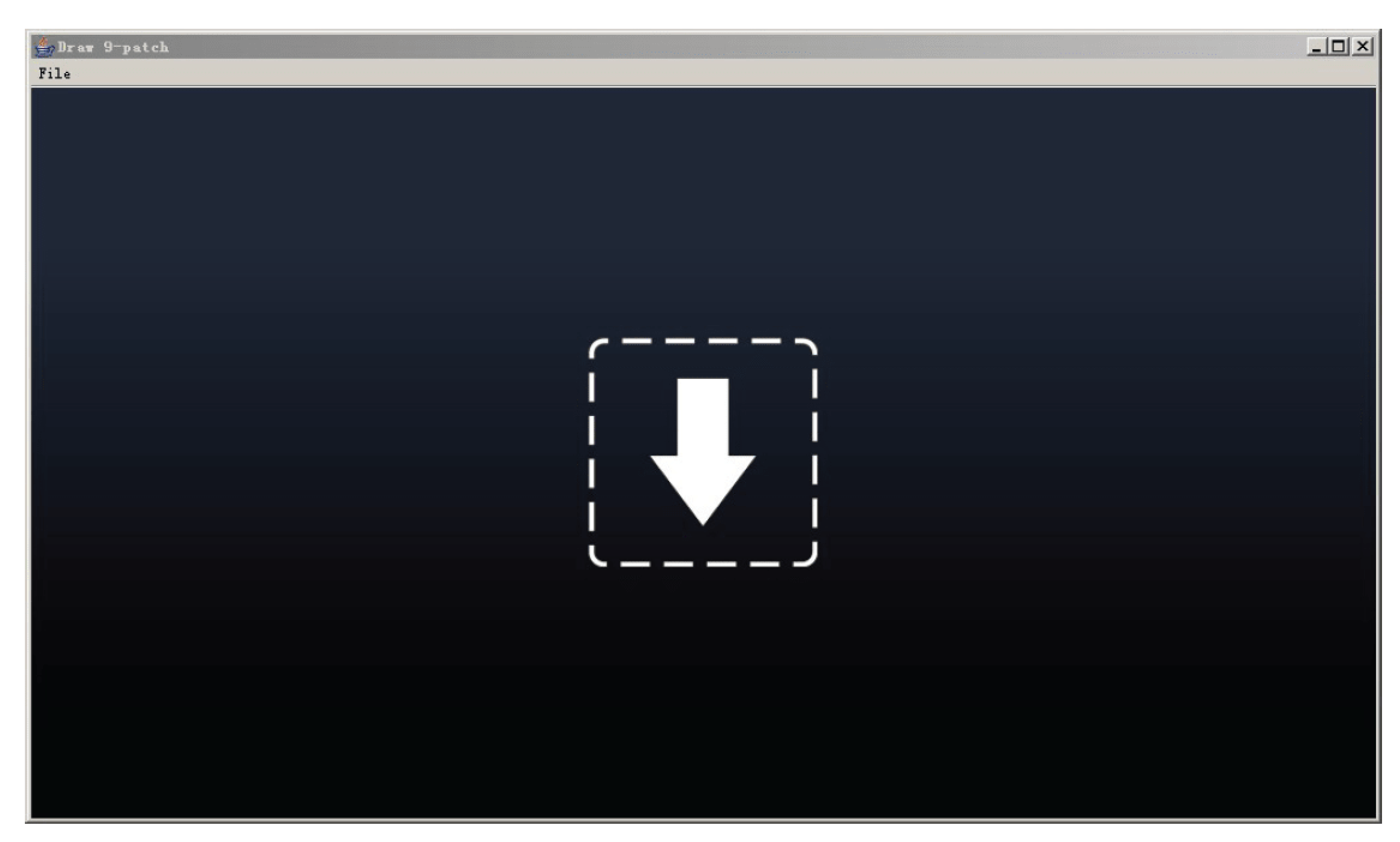


图 6.9　draw9patch 运行初始界面

（2）将一个 png 格式的图片拖入面板中，或者在 File 菜单中单击 open 9-patch，选中文件。拖入文件后界面显示如图 6.10 所示。

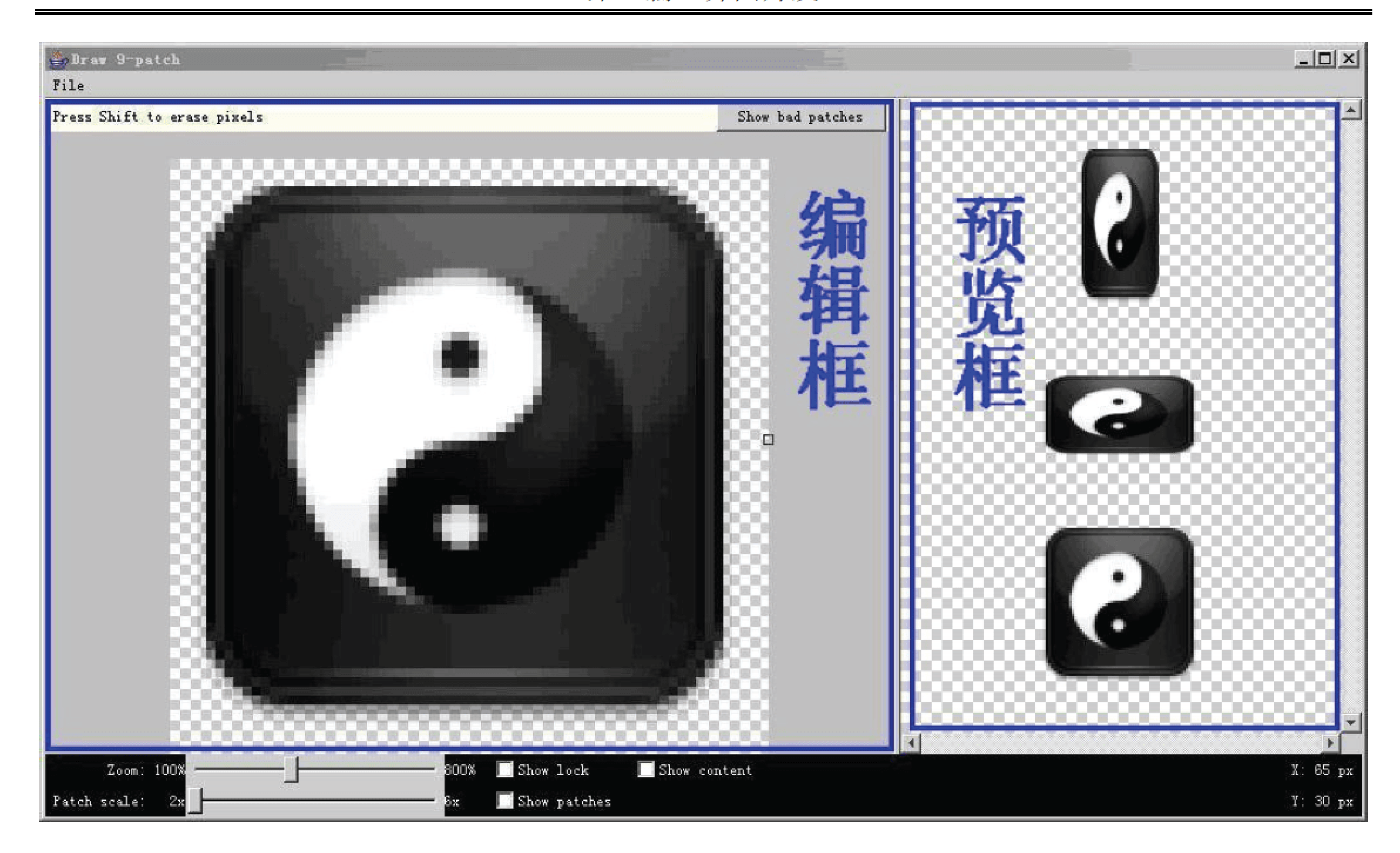


图 6.10　拖入图片后

（3）去除 Show lock 复选框，选中 Show patches 复选框。

（4）拖动 zoom 滑块放大缩小；拖动 Patch scale 滑块调整刻度比例等直到一个合适的值。

（5）沿着图像上边沿单击，设置水平“格”。效果如图 6.11 所示，其中标注“1”的两条垂直的框即为被水平拉伸的内容：

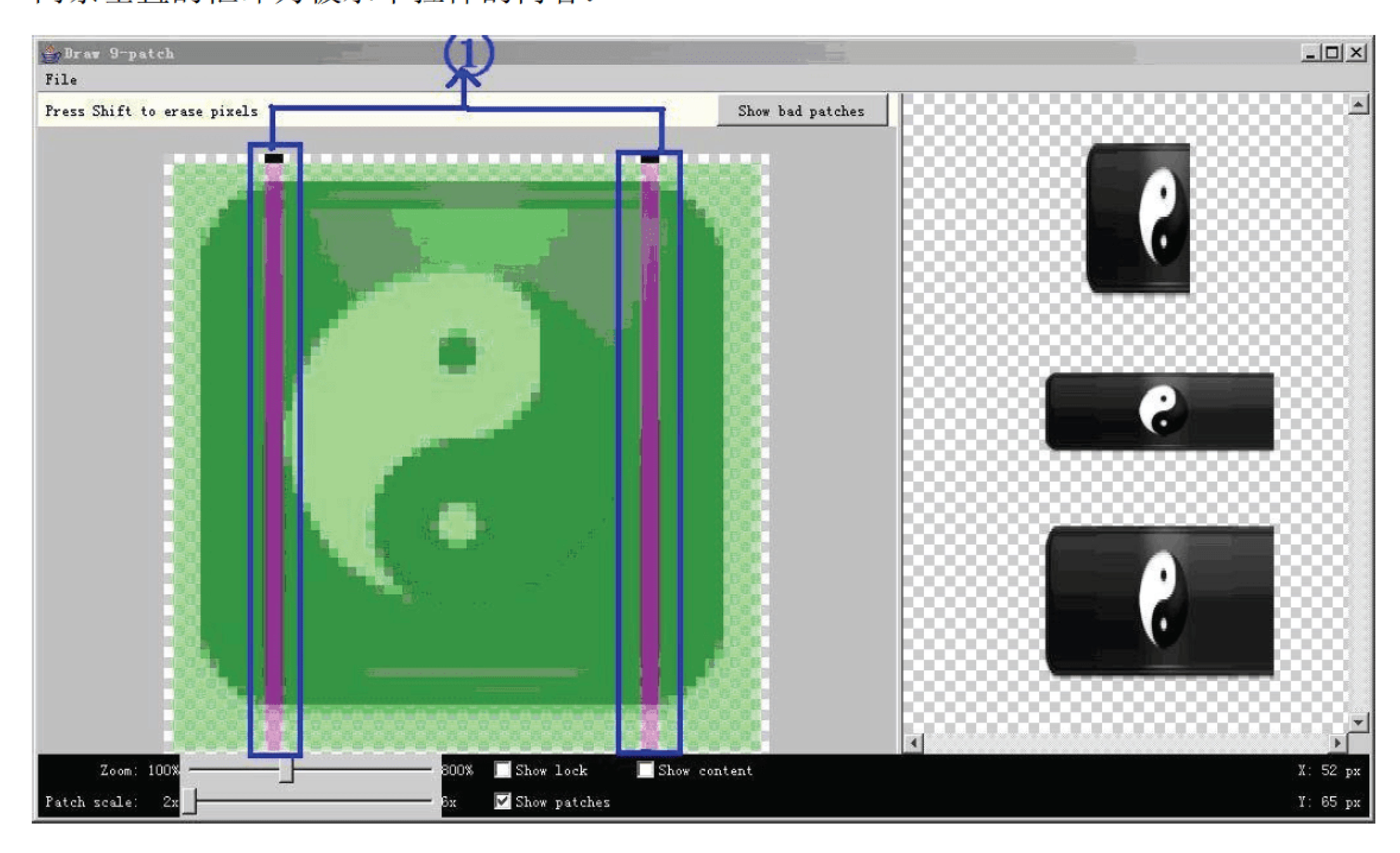


图 6.11　设置水平拉伸

（6）沿着图像左边沿单击，设置垂直“格”。效果如图 6.12 所示，其中标注“2”的两条水平的框即为被垂直拉伸的内容，标注“3”的一个矩形区域既水平拉伸又垂直拉伸。

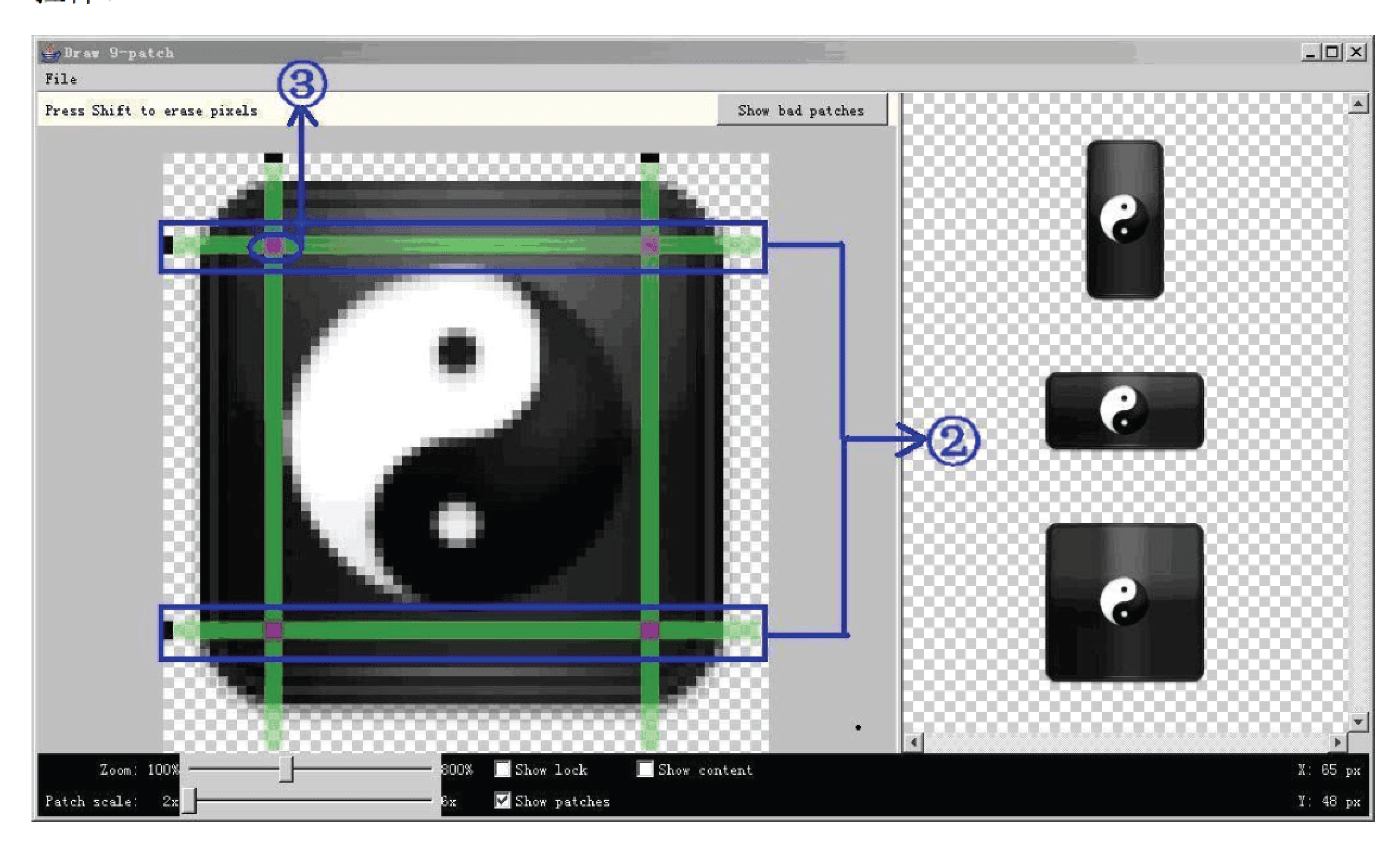


图 6.12　设置垂直拉伸

我们可以再仔细观察编辑框并截取其中一部分图片加以分析，如图 6.13 所示。将 6.14 图中的截取部分放大，观察其结构，可以得到图 6.13。

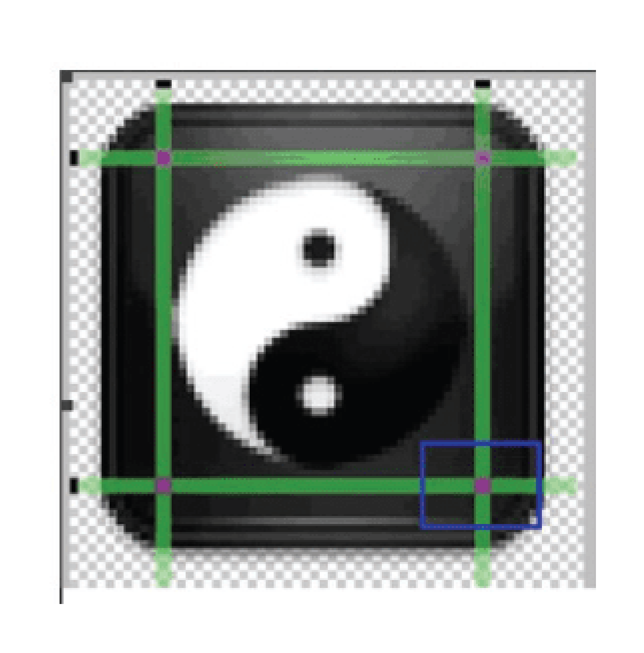


图 6.13　编辑框截取图

不拉伸	仅水平拉伸	不拉伸
仅垂直拉伸	水平拉伸，同时垂直拉伸	仅垂直拉伸
不拉伸	仅水平拉伸	不拉伸

图 6.14　截取框注释图

这也是 9.png 被称为 9 格拉伸图的原因所在了。

（7）要删除一个格只需在黑色部分右键单击或者按住 Shift 键再左键单击就可以了。

（8）最后保存图片，保存的图片就以 9.png 为后缀名了。

保存完文件我们会发现 9.png 格式的图片比之前的 png 格式图片多了几个黑点，不要担心，使用时没有问题的。事实上 9.png 格式的图片比原图片多了一圈由空白像素组成的边框。

这个时候再将图片拖入到资源中，并作为背景图片，运行一下看看效果如何吧，如图 6.15 所示。

6.2.8　使用样式

样式是一个或者多个格式化属性的集合，我们经常使用它来设置字体大小、颜色等属性。样式可以作为一个独立的属性添加到 View 的属性中，这样就大大地减少了重复的代码部分，解放了编程时程序员的劳动。样式同样由 xml 文件定义，在编译程序时被编译到程序二进制中。

需要注意的是，样式在 Android Eclipse 中不能被预览，但是不用担心，在显示时不会有问题的。

使用样式时需使用<style>标签，并在其下包含<item>选项，例如我们希望该 View 的字体大小是 20sp，颜色为红色，其语法格式如下：

图 6.15　使用 9.png 格式的图片

```
<style name="mystyle1">
       <item
name="android:textColor">#FFFF0000</item>
       <item name="android:textSize">20sp</item>
</style>
```

使用时只需将 style 做一个属性添加到 View 中就可以了。比如：

```
<TextView
    android:id="@+id/textView0"
   style="@style/mystyle1"
      />
```

这样两行属性我们就已经为该 TextView 设置了字体大小为 20sp 和颜色为红色。

这样看可能读者朋友们还没有感觉到 style 的强大之处，接下来我们可以通过一个实例完成对 style 的深入理解。依旧是彩虹和太极的实例，这里将介绍性文字的字体大小设置为 30sp、字体颜色设为白色；显示具体颜色的字体设为 20sp、字体颜色设为红色。我们看一下不用 style 时的 main.xml。

1. 不用style时

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
    android:id="@+id/textView0"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/rainbow"
   android:textSize="30sp"
   />
<TextView
    android:text="@string/red"
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:textSize="@dimen/sp"
    android:textColor="@color/red"></TextView>
<TextView
    android:text="@string/orange"
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="@dimen/sp"
    android:textColor="@color/red"></TextView>
<TextView
    android:text="@string/yellow"
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="@dimen/sp"
    android:textColor="@color/red"></TextView>
<TextView
    android:text="@string/green"
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="@dimen/sp"
    android:textColor="@color/red"></TextView>
<TextView
    android:text="@string/cyan"
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="@dimen/sp"
    android:textColor="@color/red"></TextView>
<TextView
    android:text="@string/blue"
    android:id="@+id/textView6"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="@dimen/sp"
    android:textColor="@color/red"></TextView>
<TextView
    android:text="@string/purple"
    android:id="@+id/textView7"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="@dimen/sp"
    android:textColor="@color/red"></TextView>

<TextView
    android:text="@string/taichi"
    android:id="@+id/textView8"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="30sp"></TextView>
<TextView
    android:text="@string/white"
    android:id="@+id/textView9"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="@dimen/sp"
    android:textColor="@color/red"></TextView>
<TextView
    android:text="@string/black"
```

```
    android:id="@+id/textView10"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:textSize="@dimen/sp"
    android:textColor="@color/red"></TextView>
</LinearLayout>
```

我们一共使用了整整 84 行才完成了布局文件的编写，运行后效果如图 6.16 所示。



图 6.16　运行效果

假如我们使用 style，那么首先新建一个 styles.xml 文件，添加代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="mystyle1">
       <item name="android:textColor">#FFFF0000</item>
       <item name="android:layout width">wrap content</item>
       <item name="android:layout height">wrap content</item>
       <item name="android:textSize">20sp</item>
   </style>
      <style name="mystyle2">
       <item name="android:layout width">wrap content</item>
       <item name="android:layout height">wrap content</item>
       <item name="android:textSize">30sp</item>
       <item name="android:textStyle">bold</item>
       <item name="android:textColor">#FFFFFFFF</item>
   </style>
</resources>
```

这样我们就定义了两个风格，分别是 mystyle1 和 mystyle2。接着我们重新修改 main.xml 文件。

2. 使用了style后的main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
<TextView
     android:id="@+id/textView0"
    style="@style/mystyle2"
    android:text="@string/rainbow"
    />
<TextView
     android:text="@string/red"
     android:id="@+id/textView1"
     style="@style/mystyle1"></TextView>
<TextView
     android:text="@string/orange"
     android:id="@+id/textView2"
     style="@style/mystyle1"></TextView>
<TextView
     android:text="@string/yellow"
     android:id="@+id/textView3"
     style="@style/mystyle1"></TextView>
<TextView
     android:text="@string/green"
     android:id="@+id/textView4"
     style="@style/mystyle1"></TextView>
<TextView
     android:text="@string/cyan"
     android:id="@+id/textView5"
     style="@style/mystyle1"></TextView>
<TextView
     android:text="@string/blue"
     android:id="@+id/textView6"
     style="@style/mystyle1"></TextView>
<TextView
     android:text="@string/purple"
     android:id="@+id/textView7"
     style="@style/mystyle1"></TextView>

<TextView
     android:text="@string/taichi"
     android:id="@+id/textView8"
     style="@style/mystyle2">
</TextView>
<TextView
     android:text="@string/white"
     android:id="@+id/textView9"
     style="@style/mystyle1"></TextView>
<TextView
     android:text="@string/black"
     android:id="@+id/textView10"
     style="@style/mystyle1"></TextView>
</LinearLayout>
```

这个时候我们发现，只需 54 行就完成了布局文件的编写，整整省略了 30 行代码的工作量，相当于减少了之前工作量的 35%！这还是一个简单的界面，如果是更加复杂的工程需要用到的组件更多，其节省的工作量将更加可观，提高的效率更多。运行一下，效果应该和之前一样，笔者就不再贴图了。

6.2.9　使用主题

与使用样式一样，使用主题同样需要定义一个 style 属性，不同的是样式只是针对一个组件，而主题是针对整个 Activity。style 的定义参照 6.2.8 小节，笔者不再赘述。使用主题时，需在 Activity 的属性中添加 android:theme 属性，并在属性中引用 style 资源。那么 Activity 属性又在哪里修改呢？没有错，就是 AndroidManifest.xml 文件中，修改后的注册文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.wes.resdemo"
     android:versionCode="1"
     android:versionName="1.0">
   <uses-sdk android:minSdkVersion="8" />

   <application android:icon="@drawable/icon" android:label="@string/
   app name">
      <activity android:name=".ResDemo"
             android:label="@string/app name"
             android:theme="@style/mystyle1"
             >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

   </application>
</manifest>
```

注意加粗部分，这部分就是我们需要的 xml 代码。我们仍然使用上一小节中的例子，这个时候的 main.xml 文件就更加简洁了。

1．main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
    android:id="@+id/textView0"
   style="@style/mystyle2"
   android:text="@string/rainbow"/>
<TextView
    android:text="@string/red"
    android:id="@+id/textView1" ></TextView>
<TextView
    android:text="@string/orange"
    android:id="@+id/textView2" ></TextView>
<TextView
    android:text="@string/yellow"
    android:id="@+id/textView3" ></TextView>
<TextView
    android:text="@string/green"
    android:id="@+id/textView4" ></TextView>
```

```
<TextView
    android:text="@string/cyan"
    android:id="@+id/textView5" ></TextView>
<TextView
    android:text="@string/blue"
    android:id="@+id/textView6" ></TextView>
<TextView
    android:text="@string/purple"
    android:id="@+id/textView7" ></TextView>
<TextView
    android:text="@string/taichi"
    android:id="@+id/textView8"
    style="@style/mystyle2" >
    </TextView>
<TextView
    android:text="@string/white"
    android:id="@+id/textView9" ></TextView>
<TextView
    android:text="@string/black"
    android:id="@+id/textView10" ></TextView>
</LinearLayout>
```

这样我们只使用了 42 行代码就完成了预期的效果。注意代码中加粗的部分，这里我们希望介绍性的文字与其他文字风格不一样，就可以重新制定风格。也就是说主题是可以在之后的编程过程中修改的，它并非一成不变。

6.3　小　　结

本章学习了使用 Android 中的各种资源，包括字符串、字符串数组、尺寸、颜色、图片样式、主题等。在 xml 文件中使用资源我们用@符号表示资源的引用，在 Java 代码中通过 R 文件快速找到资源。R 文件是系统自动生成的，我们不能手动去修改。本章重点在于各类组件的定义方法，难点是通过 draw9patch 工具制作 9.png 格式图片。

将本章与第 5 章结合使用，会编写出非常实用而友好的界面，同时代码结构会非常简洁。下一章我们将更系统地学习布局，以便更好地组织组件和资源，达到更加绚丽的界面效果。

第 7 章　设计界面布局

本章我们将讲解如何在 Android 中进行界面布局。通过本章的学习，读者将掌握如何使用 Android 提供的一些布局类，包括 LinearLayout、TableLayout、FrameLayout、RelativeLayout 以及 AbsoluteLayout 等。并且我们将学习一些功能与布局类类似的容器视图。

7.1　创 建 界 面

在 Android 应用中创建界面通常有两种方法，一种是使用 xml 创建布局，这在之前的范例程序中经常被使用，也许读者朋友们对其已经比较熟悉了。第二种则是在 Java 代码中实现，与使用 xml 文件相比，它更加灵活、更加“动态”，缺点则是会使代码比较混乱，不如使用 xml 文件那样结构清晰。

7.1.1　使用 xml 资源创建布局

假如读者已经仔细阅读了本书，那么应该知道，使用 xml 资源文件创建界面时，文件位于/res/layout 文件夹下。该方法是创建界面最方便也是最常用的方法，在创建时你需要为它赋予一些属性，当然在之后的程序代码中你还可以对其进行修改。

让我们来观察一个最简单的布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@String/hello"
    />
</LinearLayout>
```

相信这个布局文件大家肯定不会陌生，所有的 Android 程序在第一次被创建时都会默认拥有该布局文件。这是一个最简单的线性布局，它只包含有一个 Widget——TextView，

接下来，我们再来仔细阅读一下这段“似曾相识”的代码。

第 1 行是所有的 xml 文件头，指定了版本和编码格式。

第 2～5 行则是线性布局的节点以及属性：

第 2 行：xmlns，这是 xml 的命名空间，也就是 xml namespace 的缩写，读者不必关心。

第 3 行：方向，可以设置为横向或者纵向。

第 4 行：布局的宽度。

第 5 行：布局的高度。

接下来的代码想必读者可以自行理解了吧，这里就不再赘述。

阅读完毕后，相信大家应该能够在脑海中想象出这个布局文件的界面内容了吧，当然，千万不要忘记在 Java 代码中还要进行的一个重要步骤：

```
setContentView(R.layout.main);
```

只有执行了本代码之后，xml 资源文件中的设置才能正确显示。

7.1.2　使用代码创建布局

如果你不愿意使用 xml 来创建布局，或者某些时候，使用 xml 创建布局反而不方便，这个时候你可以选择在 Java 代码中完成布局的创建工作。当然，这最好只是特殊情况下做出的无奈选择，因为一旦使用 Java 代码进行布局，后期的维护将非常困难，而且创建时也比较困难。

接下来我们就以上一小节中的布局为例，在 Java 代码中进行创建：

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

    //创建 TextView
        TextView tv = new TextView(this);
    //设置内容
        tv.setText("HellloWorld!!");
    //设置字体大小
tv.setTextSize(50);
    //创建线性布局
        LinearLayout ll = new LinearLayout(this);
    //设置方向
        ll.setOrientation(LinearLayout.VERTICAL);
    //将 TextView 添加到线性布局中
        ll.addView(tv);
    //显示布局
        setContentView(ll);
    }
```

正如你所见，所有的组件和布局都是在 Java 代码中被动态创建，通过程序中的注释，相信大家自行阅读理解本段代码应该没有问题。运行一下程序，观察一下是不是和使用 xml 资源文件创建布局效果相同呢？事实上，运行后效果如图 7.1 所示。



图 7.1　使用代码创建布局

7.2　使用布局类

Android SDK 为我们提供了 5 个布局类，它们是：线性布局（LinearLayotu）、绝对布局（AbsoluteLayout）、表格布局（TableLayout）、关系布局（RelativeLayout）、框架布局（FrameLayout）。本节将逐一讲解这些类的使用方法和技巧。

7.2.1　使用绝对布局

绝对布局（AbsoluteLayout）视图是指为该布局内的所有子视图指定一个绝对的坐标。前文已经提过每个视图最终都是一块矩形的区域，不同的组件对该区域进行不同的渲染。而绝对布局为每个视图指定的坐标点就是以矩形区域的左上角为基准，坐标的形式是（x 轴坐标，y 轴坐标）。

这样精确地指定每一个坐标的位置似乎可以很准确地得到我们希望得到的最终界面。但实际上绝对视图在真正的开发中是很少使用的，因为它的可移植性太差，开发过程中针对不同的硬件需要修改相应的参数，非常的麻烦。所以官方给出的建议是在开发中尽量不要使用绝对布局，使用其他的布局方式同样可以取得最终的效果。

那为什么我们还要讲解这个布局类呢？因为到目前为止，绝对布局仍然可以正常使用，而且在开发一些需要精确到像素级别的视图时还需要用到它。总结一下就是一句话：艺多不压身，起码这也是一条解决问题的途径，当然是最后一条了，有点背水一战的意味。

言归正传，我们来观察具体的绝对布局的使用方法。

1. 通过xml资源创建绝对视图

首先请看以下代码段：

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
     <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="坐标：(0,0)"
    android:layout_x="0px"
    android:layout_y="0px"
    android:textSize="25sp"
    />
    <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="坐标：(150,20)"
    android:layout_x="150px"
    android:layout_y="20px"
    />
    <AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="50px"
    android:layout_y="100px"
    />
    <DigitalClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="200px"
    android:layout_y="300px"
    />
</AbsoluteLayout>
```

通过阅读代码，我们可以很清晰地发现，绝对布局就是在 AbsoluteLayout 根节点下将若干视图作为子节点，并赋予各自的 layout_x 和 layout_y 属性，就完成了界面的搭建工作。

运行后，界面显示效果如图 7.2 所示。

2. 通过代码创建绝对布局

在代码中实现动态布局会比较麻烦一些，要使用代码实现绝对布局需要以下 5 个步骤：

（1）创建需要显示的组件对象。

（2）创建布局参数对象。

（3）创建绝对布局对象。

（4）将组件对象添加到布局对象中，并赋予其相应的布局参数。

（5）使用 setContentView()方法将布局显示。

代码段如下所示：



图 7.2　绝对布局

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //创建各个视图
        TextView tv = new TextView(this);
        EditText et = new EditText(this);
        AnalogClock ac = new AnalogClock(this);
        AnalogClock ac2 = new AnalogClock(this);
        //创建绝对布局
         AbsoluteLayout al = new AbsoluteLayout(this);
        //创建布局参数
       LayoutParams params1 = new LayoutParams(ViewGroup.LayoutParams.
       WRAP CONTENT,
ViewGroup.LayoutParams.WRAP CONTENT,0,0);
        //添加 TextView
        tv.setTextSize(25);
        tv.setText("参数设置不当会造成重叠");
        al.addView(tv,params1);
        //添加 EditText
        LayoutParams params2 = new LayoutParams(ViewGroup.LayoutParams.
        WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT,50,50);
        et.setText("(20,100)");
        al.addView(et,params2);
        //添加 AnalogClock
        LayoutParams params3 = new LayoutParams(ViewGroup.LayoutParams.
        WRAP CONTENT,ViewGroup.LayoutParams.WRAP CONTENT,50,100);
        al.addView(ac,params3);

        //添加 DigitalClock
        LayoutParams params4 = new LayoutParams(ViewGroup.LayoutParams.
        WRAP_CONTENT,ViewGroup.LayoutParams.WRAP_CONTENT,
```

```
                                     150,200);
        al.addView(ac2,params4);

        //将布局显示
        setContentView(al);
    }
```

运行以上代码在模拟器中我们可以得到如图 7.3 所示的界面，而在真机测试时得到的界面却如图 7.4 所示。



图 7.3　模拟器中的显示

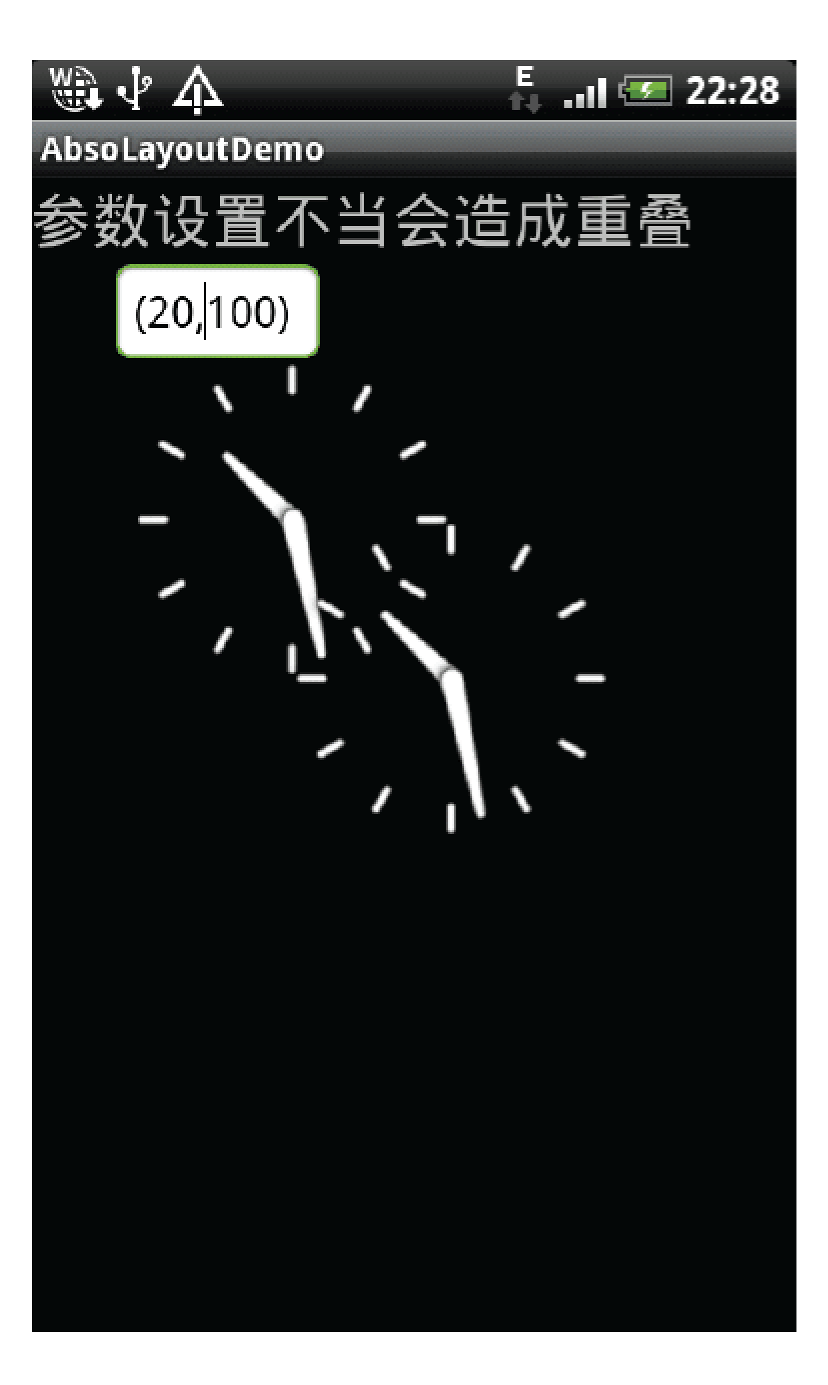


图 7.4　真机显示

通过比较两者的显示，我们很容易就发现了为什么官方不建议我们在开发时使用绝对布局了，因为同样的代码在不同的硬件上显示的效果完全不同。也许在开发时你的界面设计的完美无瑕，可是运行在其他的手机屏幕上就是一团糟。图 7.4 中两个时钟居然重叠在一起了！

造成这些的原因就是每类手机硬件的像素密度都不同，不同的像素密度就会造成不同的显示效果。所以下一小节我们将学习一个比较实用的布局类——线性布局类。

7.2.2　使用线性布局

线性布局是开发人员在开发中使用最多的一类布局，甚至在 Android 新建工程时默认的布局都是 LinearLayout。线性布局的作用是将所有的子视图按照横向或者纵向有序地排列。这里不得不提到线性布局特有的一个属性——android：orientation，该属性的作用是指定本线性布局下的子视图排列方向，如果设置为“horizontal”则表示水平，方向为从左向右；若设置为“vertical”则表示垂直，方向为从上向下。将多个线性布局嵌套可以完成大部分希望实现的效果。

1．使用xml编写线性布局

接下来我们依然阅读一段 LinearLayout 的 xml 代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent"
   >
    <TextView
   android:layout width="fill parent"
   android:layout height="wrap content"
   android:text="水平方向"
   />
   <LinearLayout
   android:orientation="horizontal"
   android:layout width="wrap content"
   android:layout height="wrap content"
   >
       <ImageView
       android:layout width="wrap content"
       android:layout height="wrap content"
       android:src="@drawable/right"
       />
         <ImageView
       android:layout width="wrap content"
       android:layout height="wrap content"
       android:src="@drawable/right"
       />
....../省略部分视图
   </LinearLayout>
   <TextView
   android:layout width="wrap content"
   android:layout height="wrap content"
   android:text="垂直方向"
   />
   <LinearLayout
   android:orientation="vertical"
   android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
   >
       <ImageView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:src="@drawable/down"
       />
        <ImageView
       android:layout width="wrap content"
       android:layout height="wrap content"
       android:src="@drawable/down"
       />
      ....../省略部分视图
   </LinearLayout>
</LinearLayout>
```

其中的 drawable/down 和 drawable/right 图片分别如图 7.5 和图 7.6 所示。



图 7.5　down



图 7.6　right

让我们分析一下这段代码，代码中总体结构为：

在一个整体的垂直线性布局中有 4 个子视图，它们从上到下依次为 TextView、LinearLayout、TextView、LinearLayout，接着在子视图的第一个 LinearLayout 中，从左向右排列了一排 ImageView，第二个 LinearLayout 中，从上到下排列了一列 ImageView。如果你愿意，还可以继续向下层嵌套，当然最好不要嵌套太深的层数，因为这会大大地降低显示效率。其框架结构如图 7.7 所示。

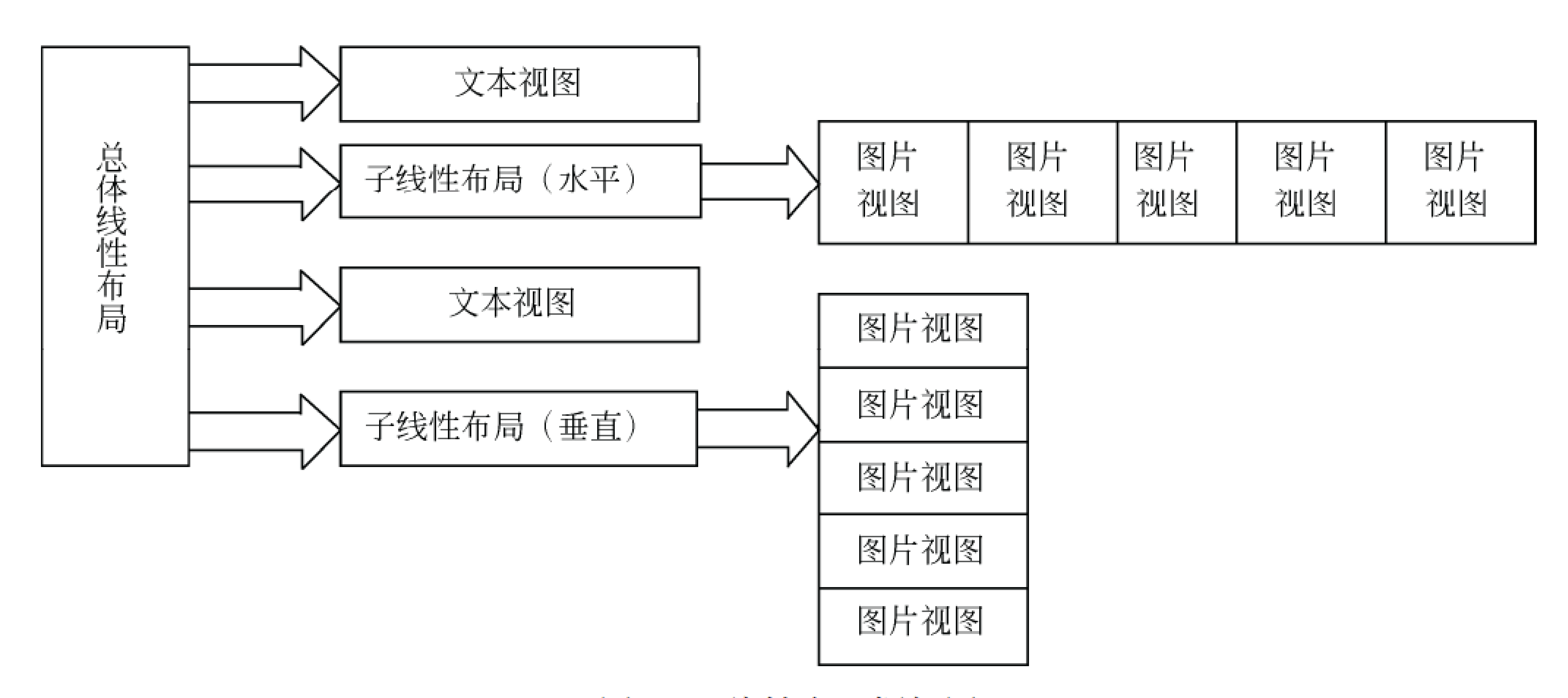


图 7.7　线性布局框架图

理解了本段代码的框架结构后我们再运行代码，看看效果是不是和我们希望的一样，效果如图 7.8 所示。

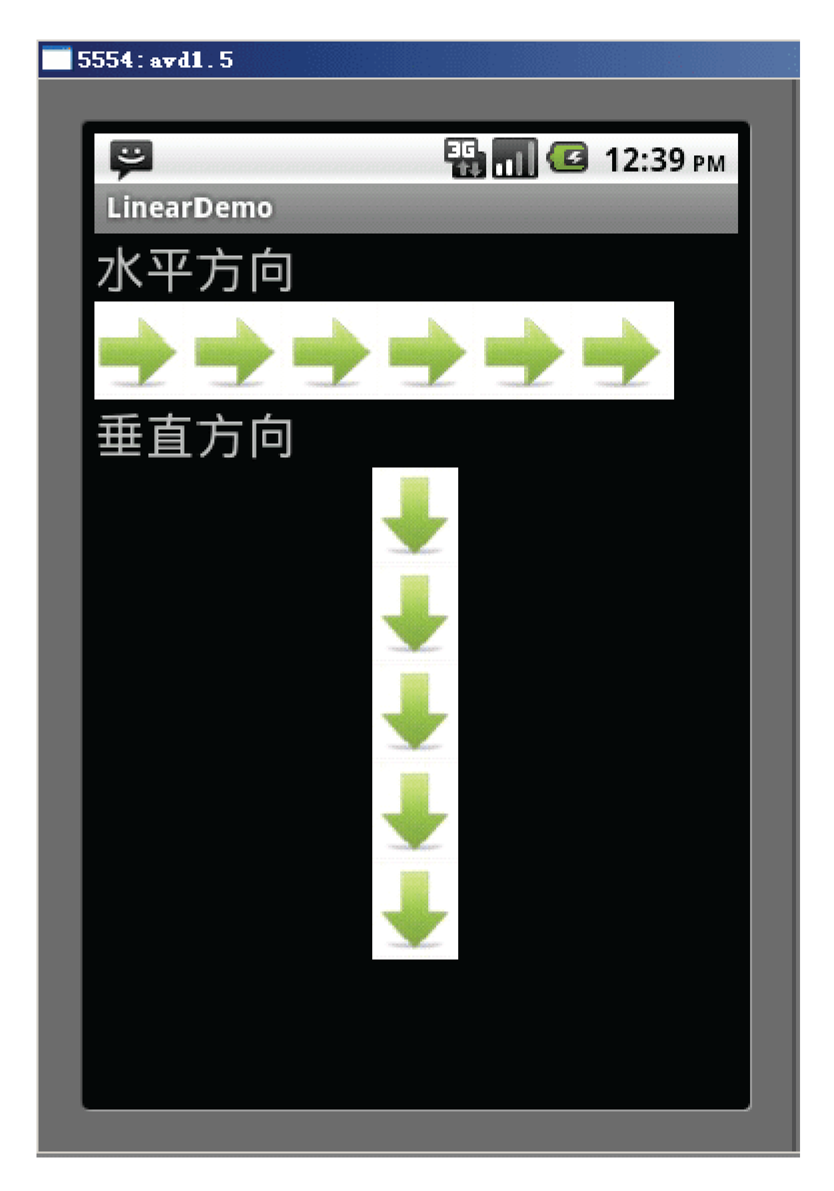


图 7.8　线性布局

2. 使用代码编写线性布局

使用 Java 代码编写线性布局会比较麻烦，而且它们的层级结构会显得没有 xml 代码那么清晰，后期修改代码时，包括改变参数时都会需要更多的工作量。

接下来我们来完成一个与上例类似地线性布局示例：

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
//创建总体的线性布局
        LinearLayout ll = new LinearLayout(this);
        ll.setOrientation(LinearLayout.VERTICAL);
        //创建第一个 TextView
        TextView tv_1 = new TextView(this);
        tv_1.setText("垂直");
        tv_1.setTextSize(25);
        //创建子线性布局 ll_1
        LinearLayout ll_1 = new LinearLayout(this);
        LinearLayout.LayoutParams params = new LinearLayout.LayoutParams
        (LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        ll_1.setOrientation(LinearLayout.VERTICAL);
        ll_1.setLayoutParams(params);
        //创建子布局中的 ImageView
ImageView iv1 = new ImageView(this);
        iv1.setBackgroundResource(R.drawable.down);
        ImageView iv2 = new ImageView(this);
        iv2.setBackgroundResource(R.drawable.down);
        ImageView iv3 = new ImageView(this);
        iv3.setBackgroundResource(R.drawable.down);
        ImageView iv4 = new ImageView(this);
        iv4.setBackgroundResource(R.drawable.down);
        ImageView iv5 = new ImageView(this);
        iv5.setBackgroundResource(R.drawable.down);
       //将 ImageView 添加到 ll_1 线性布局中
 ll_1.addView(iv1);
        ll_1.addView(iv2);
        ll_1.addView(iv3);
        ll_1.addView(iv4);
        ll_1.addView(iv5);
        //创建第二个 TextView
        TextView tv_2 = new TextView(this);
        tv_2.setText("水平");
        tv_2.setTextSize(25);
        //创建第二个子线性布局 ll_2
        LinearLayout ll_2 = new LinearLayout(this);
        ll_2.setOrientation(LinearLayout.HORIZONTAL);
        ImageView iv6 = new ImageView(this);
        iv6.setBackgroundResource(R.drawable.right);
        ImageView iv7 = new ImageView(this);
        iv7.setBackgroundResource(R.drawable.right);
        ImageView iv8 = new ImageView(this);
        iv8.setBackgroundResource(R.drawable.right);
        ImageView iv9 = new ImageView(this);
        iv9.setBackgroundResource(R.drawable.right);
        ImageView iv10 = new ImageView(this);
        iv10.setBackgroundResource(R.drawable.right);
         //将 ImageView 添加到 ll_2 布局中
        ll_2.addView(iv6);
```

```
        ll 2.addView(iv7);
        ll 2.addView(iv8);
        ll 2.addView(iv9);
        ll 2.addView(iv10);
        //将两个 TextView 和两个 LinearLayout 都添加到总体的线性布局中
        ll.addView(tv_1);
        ll.addView(ll_1);
        ll.addView(tv_2);
        ll.addView(ll_2);
        setContentView(ll);
    }
```

第一遍阅读本段代码也许会比较混乱，这个时候我们不能看局部的代码而应该从“俯视”的角度来观察它的总体结构。从命名上你也会发现它们的层级关系与图 7.7 是完全一样的，不同的只是排列的顺序而已。运行代码，效果如图 7.9 所示。

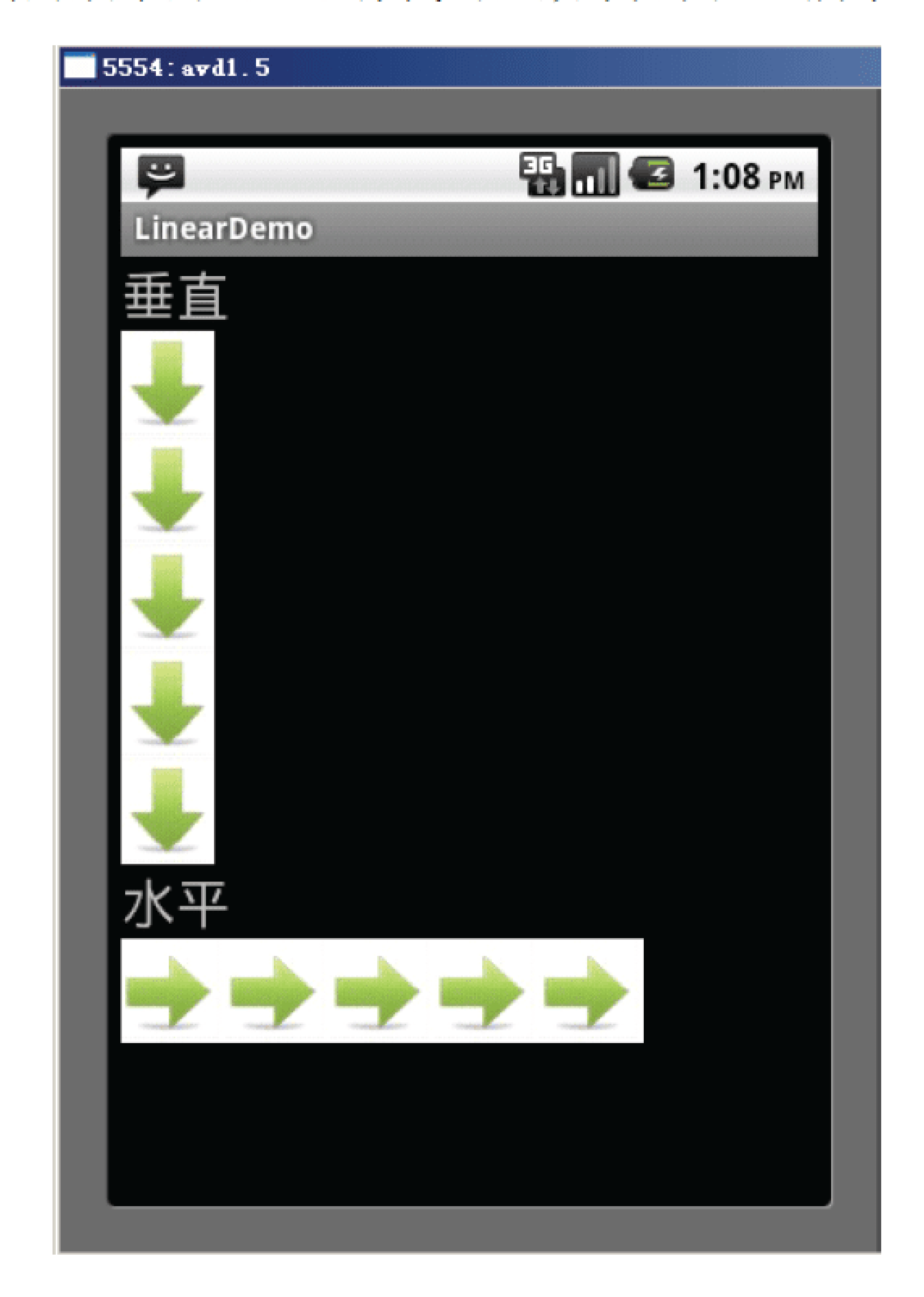


图 7.9　Java 代码编写线性布局

7.2.3　使用框架布局

由上一小节我们知道，线性布局是将所有的子视图按照一定的方向有序排列，那么如果我们希望某些子视图能够堆叠在一起以产生某些“特殊”的效果呢？例如，我们要实现如下的效果：给出一幅图片，在图片上做出一些标注。按照目前我们学习的布局好像只有绝对布局才能达到效果，可是之前笔者又建议大家最好不要使用绝对布局，怎么办呢？

这个时候我们就需要使用框架布局了。框架视图常被用来显示一堆子视图，每个视图都会默认地以屏幕左上角为参照进行绘制，第一个“画”的视图会被“画”在最底层，第

二个视图会在上一层，依此类推，最后一个视图会被“画”在最顶层。

框架布局非常简单而搞笑，如果使用层级视图工具（Hierarchy Viewer tool）你会发现所有的布局都是在一个总体的框架布局中。事实上，我们手机的主界面（Home 界面）就是使用的框架视图，每个小应用都是一个子视图。

1．使用xml文件创建框架视图

首先我们准备一张图片，如图 7.10 所示。



图 7.10　湖面

接着开始我们的框架布局的编写：

```
<?xml version="1.0" encoding="utf-8"?>
 <FrameLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent">
 <ImageView
android:layout_height="fill_parent"
 android:layout_width="fill_parent"
 android:src="@drawable/bg"
 android:scaleType="matrix"
 ></ImageView>
 <TextView
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:textColor="#000"
 android:textSize="40dp"
 android:text="天空"
 android:gravity="center"
 />
 <TextView
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="倒影"
 android:layout_gravity="bottom"
 android:gravity="center"
 android:textColor="#000"
```

```
android:textSize="40dp" />
</FrameLayout>
```

通过阅读代码，我们知道最底层的视图是一个图片视图 ImageView，然后是一个文本视图，最高层还是一个文本视图，不过两个文本视图一个默认在顶部，还有一个通过 android:layout_gravity="bottom"将其指定为在父视图的底部，与此同时它们都通过：

```
android:gravity="center"
```

将文字定位在了本视图的中间。运行程序，我们会看到如图 7.11 所示的效果。



图 7.11　框架视图

2. 在Java代码中编写框架视图

在 Java 中编写框架布局的代码与编写线性布局类似，需要使用一些 LayoutParams 来设置属性：

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.main);
        //创建图片视图并设置属性
        ImageView iv = new ImageView(this);
        iv.setImageResource(R.drawable.bg);
        LayoutParams params1 = new LayoutParams(LayoutParams.FILL_PARENT,
        LayoutParams.FILL_PARENT);
        iv.setLayoutParams(params1);
        iv.setScaleType(ScaleType.MATRIX);
        //创建第一个文本视图
```

```
        TextView tv1 = new TextView(this);
        LayoutParams params2 = new LayoutParams(LayoutParams.FILL_PARENT,
        LayoutParams.WRAP_CONTENT, Gravity.TOP);
        tv1.setLayoutParams(params2);
        tv1.setText("天空");
        tv1.setTextColor(Color.BLACK);
        tv1.setTextSize(40);
        tv1.setGravity(Gravity.CENTER);
        //创建第二个文本视图
        TextView tv2 = new TextView(this);
        LayoutParams params3 = new LayoutParams(LayoutParams.FILL_PARENT,
        LayoutParams.WRAP_CONTENT, Gravity.BOTTOM);
        tv2.setLayoutParams(params3);
        tv2.setText("倒影");
        tv2.setTextColor(Color.BLACK);
        tv2.setTextSize(40);
        tv2.setGravity(Gravity.CENTER);
        //创建框架布局
        FrameLayout fl = new FrameLayout(this);
        fl.addView(iv);
        fl.addView(tv1);
        fl.addView(tv2);
        setContentView(fl);
    }
```

通过代码中的注释，读者朋友们应该能够把握本段代码段的结构了，这里就不再赘述，唯一值得一提的就是 FrameLayout.LayoutParams 类的使用，其中的 Gravity 的使用可以灵活地创建各种效果。

运行以上代码段，最后展示的效果与 xml 资源文件的布局方法是完全一样的。

7.2.4　使用表格布局

表格视图有些类似于我们平时使用的 Excel 表格，它将包含的子视图放在一个个单元格内，我们可以控制布局的行数以及列数。使用 TableLayout 可以很方便地构建计算器、拨号器等使用界面。

1．使用xml文件创建表格布局

TableLayout 与 LinearLayout 相似，添加到表格布局中的每个 TableRow 中的视图按照添加的顺序从上到下依次排列，然后添加到每个 TableRow 中的子视图按照添加顺序从左至右排列。接下来我们就完成一个使用表格布局构建的拨号器，请阅读以下代码段：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:stretchColumns="*"
   android:shrinkColumns="1,2"
   >

    <TableRow>
        <Button
        android:layout width="wrap content"
        android:layout height="wrap content"
        android:text="1"
```

```
        />
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="2"
        />
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="3"
        />
    </TableRow>
    <TableRow>
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="4"
        />
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="5"
        />
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="6"
        />
    </TableRow>
        <TableRow>
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="7"
        />
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="8"
        />
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="9"
        />
    </TableRow>
    <TableRow>
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="*"
        />
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="0"
        />
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="#"
```

```
            />
        </TableRow>
        <View
        android:layout height="3dp"
        android:background="#FF909090"
        />
        <TableRow>
            <Button
            android:layout width="wrap content"
            android:layout height="wrap content"
            android:text="拨打"
            android:layout_span="3"
            android:gravity="center"
            />
        </TableRow>
</TableLayout>
```

注意到本段表格视图中一共出现了 5 个 TableRow，所以可以预见最后的界面必定有 5 行。接着我们观察到前 4 个 TableRow 都拥有 3 个 Button，换言之也就是每行有 3 列。这个时候我们发现最后一行中只有 1 个 Button，那么是不是就只有一列了呢？不是的，表格视图是按照所有的行数中列最多的那一行进行分列的。也就是说即使你最后一行只有一个 Button，但是该行依然有 3 列，这个 Button 默认显示在第一列。

这里我们使用了 android:layout_span="3"属性，它的作用是合并 3 个单元格，这样该行就只有一列了。当然，我们也可以使用 android：layout_column 来指定该 Button 位于 3 列中的第几列。

其中加粗部分值得一提，因为这是表格视图特有的属性：

```
android:stretchColumns="*"
```

其功能为所有的列都可以伸展，以适应显示：

```
android:shrinkColumns="1,2"
```

其功能为 1，2 列可以收缩以适应显示。运行后，显示效果如图 7.12 所示。



图 7.12　表格视图

2．使用Java代码编写表格视图

使用 Java 代码编写表格视图时，需要使用两列参数设置，分别是 TableLayout.Layout Params 和 TableRow.LayoutParams，使用时需要注意。

```
@Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
                                       //创建需要使用的按钮
      Button b0 = new Button(this);
      b0.setText("0");
      Button b1 = new Button(this);
      b1.setText("1");
      ......                           //此处省略 2～8 按钮的创建工作
      b9.setText("9");
      Button b10 = new Button(this);
      b10.setText("*");
      Button b11 = new Button(this);
      b11.setText("#");
      Button b12 = new Button(this);
      b12.setText("拨打");
                                       //创建需要使用的行，并将按钮添加到行中
      TableRow tr1 = new TableRow(this);              //第一行
      tr1.addView(b1);
      tr1.addView(b2);
      tr1.addView(b3);
      TableRow tr2 = new TableRow(this);              //第二行
      tr2.addView(b4);
      tr2.addView(b5);
      tr2.addView(b6);
      TableRow tr3 = new TableRow(this);              //第三行
      tr3.addView(b7);
      tr3.addView(b8);
      tr3.addView(b9);
      TableRow tr4 = new TableRow(this);              //第四行
      tr4.addView(b10);
      tr4.addView(b0);
      tr4.addView(b11);
      TableRow tr5 = new TableRow(this);              //第五行
                                                      //行参数
      TableRow.LayoutParams lp = new TableRow.LayoutParams();
      lp.span = 3;                                    //设置合并 3 列
      tr5.addView(b12,lp);                            //添加时需带入参数

      TableLayout tl = new TableLayout(this);
      tl.setStretchAllColumns(true);
      tl.addView(tr1);
      tl.addView(tr2);
      tl.addView(tr3);
      tl.addView(tr4);
      tl.addView(tr5);

      setContentView(tl);
   }
```

再次强调，要合并列时需要使用的参数是 TableRow.LayoutParams，而不是 TableLayout.LayoutParams，具体的方法是将该参数的 span 字段设置一个大于一的整数。然

后将参数和子视图一并添加到 TableRow 中就可以了。其中：

```
tl.setStretchAllColumns(true);
```

该方法的作用是设置所有列都可以伸展，当然你也可以指定某些列可伸展，此时就需要使用方法：

```
setColumnStretchable(int columnIndex, boolean isStretchable)
```

该方法在本文中没有使用，如果读者有兴趣可以自行尝试使用。

7.2.5　使用关系布局

关系布局可以通过指定视图与其他视图的关系来确定其自身的位置，如位于某视图的上方、下方、左方、右方等，还可以指定它位于父布局的中间、右对齐、左对齐等等。这样可以避免使用多重布局，有效地提高了效率。

接下来，我们一起来完成一个有趣的实例，通过关系布局完成一个经典的太极八卦图，首先我们要准备一些图片，分别表示八卦的各个方位，如图 7.13 所示。

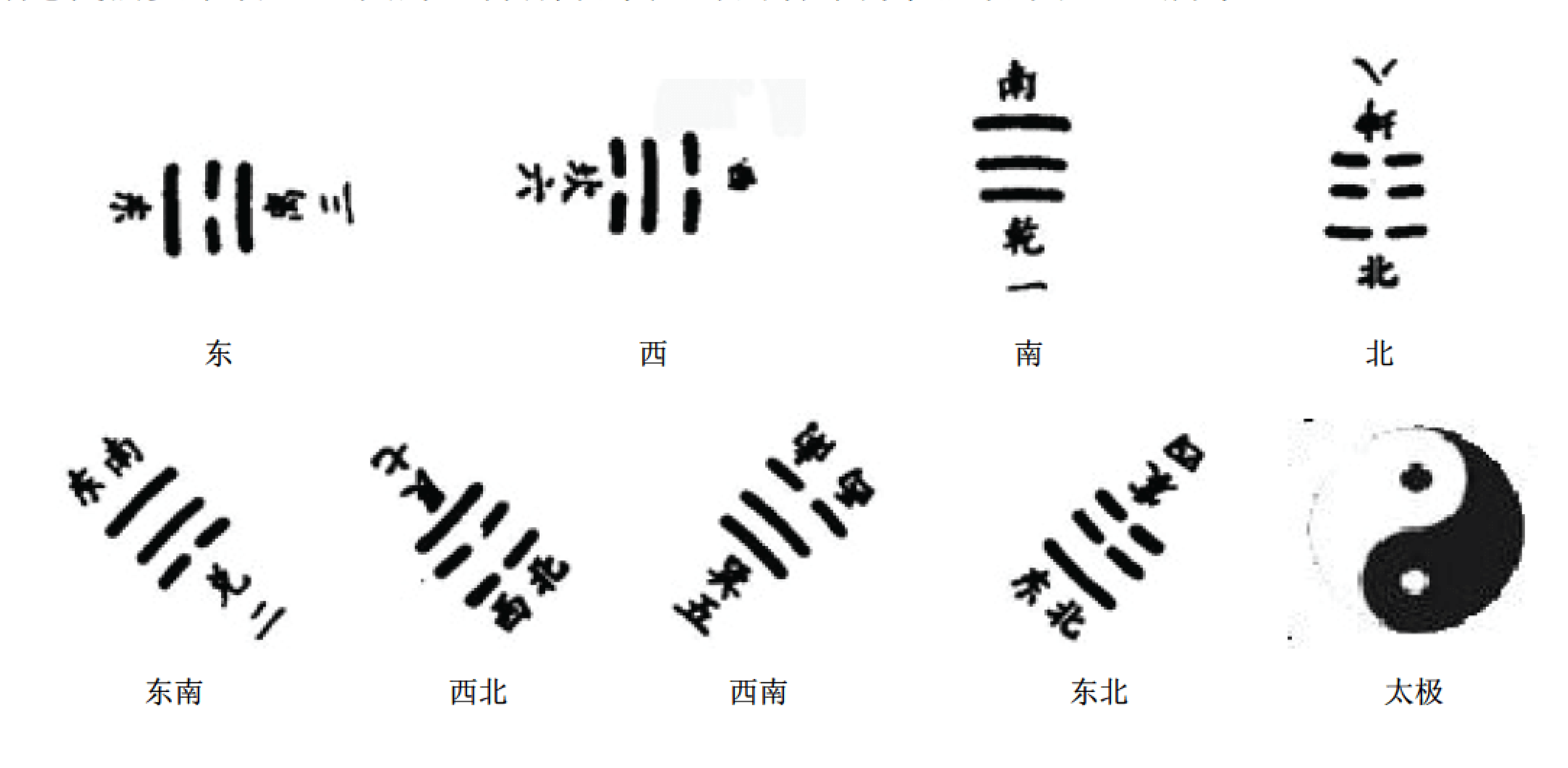


图 7.13　八卦的各个部分

接下来让我们通过关系布局将这些杂乱的图片组装起来吧！

1. 使用xml代码创建关系布局

让我们一起来阅读以下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout width="fill parent"
   android:layout height="fill parent"
   android:background="#fff"
   >
    <ImageView
```

```
 android:id="@+id/view0"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/v0"
android:layout_centerInParent="true"          //将太极定位在屏幕的中间
/>
<ImageView
android:id="@+id/view1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/v1"
android:layout_centerHorizontal="true"        //定位在屏幕的水平方向中间部分
android:layout_alignParentTop="true"          //贴着屏幕上方
/>
<ImageView
android:id="@+id/view2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/v2"
android:layout_centerHorizontal="true"        //定位在屏幕的水平方向中间部分
android:layout_alignParentBottom="true"       //贴着屏幕底部
/>
<ImageView
android:id="@+id/view3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/v3"
android:layout_centerVertical="true"          //定位在屏幕的垂直方向中间部分
android:layout_alignParentLeft="true"         //贴着屏幕左部
/>
<ImageView
android:id="@+id/view4"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/v4"
android:layout_centerVertical="true"          //定位在屏幕的垂直方向中间部分
android:layout_alignParentRight="true"        //贴着屏幕右部
/>
<ImageView
android:id="@+id/view5"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/v5"
 android:layout_alignParentRight="true"       //贴着屏幕右部
 android:paddingTop="40dp"                    //距顶部 40dp 距离
/>
<ImageView
android:id="@+id/view6"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/v6"
android:layout_alignParentLeft="true"         //贴着屏幕左边
 android:paddingTop="40dp"                    //距顶部 40dp 距离
/>
<ImageView
android:id="@+id/view7"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/v7"
```

```
    android:layout_alignRight="@id/view6"        //与 id 为 view6 的组件右对齐
     android:layout_below="@id/view0"            //在 id 为 view0 的组件下方
     android:paddingTop="20dp"                   //距离顶部 20dp 距离
    />
      <ImageView
    android:id="@+id/view8"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:src="@drawable/v8"
    android:layout_alignRight="@id/view5"        //与 id 为 view5 的组件右对齐
     android:layout_alignBottom="@id/view7"      //与 id 为 view7 的组件底部对齐
    />

</RelativeLayout>
```

通过阅读程序中的注释，相信大家应该可以独立完成阅读和理解工作，接下来我们就一起来看看运行之后的效果图吧！如图 7.14 所示。



图 7.14　八卦图（由外向内看）

接下来让我们总结一下关系布局中需要使用的属性，如表 7-1 所示。

表 7-1　布局中的属性

属　　性	描　　述	值
android:layout_centerInParent	在父视图中正中心	true/false
android:layout_centerHorizontal	在父视图的水平中心线	true/false
android:layout_centerVertical	在父视图的垂直中心线	true/false
android:layout_alignParentTop	紧贴父视图顶部	true/false
android:layout_alignParentBottom	紧贴父视图底部	true/false
android:layout_alignParentLeft	紧贴父视图左部	true/false

续表

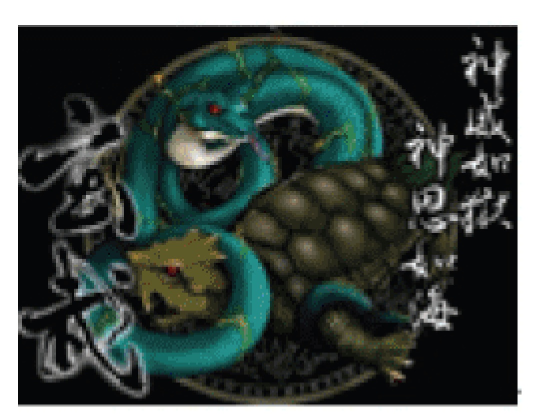
属　　性	描　　述	值
android:layout_alignParentRight	紧贴父视图右部	true/false
android:layout_alignTop	与指定视图顶部对齐	视图 ID，如“@id/***”
android:layout_alignBottom	与指定视图底部对齐	视图 ID，如“@id/***”
android:layout_alignLeft	与指定视图左部对齐	视图 ID，如“@id/***”
android:layout_alignRight	与指定视图右部对齐	视图 ID，如“@id/***”
android:layout_above	在指定视图上方	视图 ID，如“@id/***”
android:layout_below	在指定视图下方	视图 ID，如“@id/***”
android:layout_toLeft	在指定视图左方	视图 ID，如“@id/***”
android:layout_toRight	在指定视图右方	视图 ID，如“@id/***”

2．使用Java代码创建关系布局

完成了八卦图之后，让我们尝试使用 Java 代码直接编写界面，并完成一张四神兽图。首先依然准备 4 张绚丽的图片，如图 7.15 所示。









组图 7.15　中国四神兽

接下来让我们一起把这 4 个神兽按照它们应该守护的方向（东——青龙，西——白虎，南——朱雀，北——玄武）组织起来。

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //创建 4 个 ImageView 对象并设置图片
        ImageView v1 = new ImageView(this);
        v1.setBackgroundResource(R.drawable.xuanwu);
        ImageView v2 = new ImageView(this);
        v2.setBackgroundResource(R.drawable.zhuque);
        ImageView v3 = new ImageView(this);
        v3.setBackgroundResource(R.drawable.baihu);
        ImageView v4 = new ImageView(this);
        v4.setBackgroundResource(R.drawable.qinglong);
        //创建关系布局对象
        RelativeLayout rl = new RelativeLayout(this);
        //创建关系布局参数
        RelativeLayout.LayoutParams lp1 = new RelativeLayout.LayoutParams
        (ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT);
        lp1.addRule(RelativeLayout.ALIGN_PARENT_TOP);   //添加条件，上对齐
        lp1.addRule(RelativeLayout.CENTER_HORIZONTAL,
RelativeLayout.TRUE); //添加条件，在水平中线
        rl.addView(v1, lp1);

        RelativeLayout.LayoutParams lp2 = new RelativeLayout.LayoutParams
```

```
        (ViewGroup.LayoutParams.WRAP CONTENT,
ViewGroup.LayoutParams.WRAP CONTENT);
        lp2.addRule(RelativeLayout.ALIGN PARENT BOTTOM);//添加条件，下对齐
        lp2.addRule(RelativeLayout.CENTER_HORIZONTAL,
RelativeLayout.TRUE);                                //水平中线
        rl.addView(v2, lp2);

        RelativeLayout.LayoutParams lp3 = new RelativeLayout.LayoutParams
        (ViewGroup.LayoutParams.WRAP CONTENT,
ViewGroup.LayoutParams.WRAP CONTENT);
        lp3.addRule(RelativeLayout.ALIGN PARENT LEFT);  //添加条件，左对齐
        lp3.addRule(RelativeLayout.CENTER_VERTICAL, RelativeLayout.TRUE);
                                                        //垂直中线
        rl.addView(v3, lp3);

        RelativeLayout.LayoutParams lp4 = new RelativeLayout.LayoutParams
        (ViewGroup.LayoutParams.WRAP CONTENT,
ViewGroup.LayoutParams.WRAP CONTENT);
        lp4.addRule(RelativeLayout.ALIGN_PARENT_RIGHT); //添加条件，右对齐
        lp4.addRule(RelativeLayout.CENTER_VERTICAL, RelativeLayout.TRUE);
                                                        //垂直中线
        rl.addView(v4, lp4);
                                                        //将布局显示到界面
        setContentView(rl);
}
```

注意这里在为视图指定位置时使用参数 RelativeLayout.LayoutParams，要指定关系需要使用 LayoutParams.addRule()方法，具体使用方法可以参照以上范例。

运行代码，我们会看到如图 7.16 所示的效果。

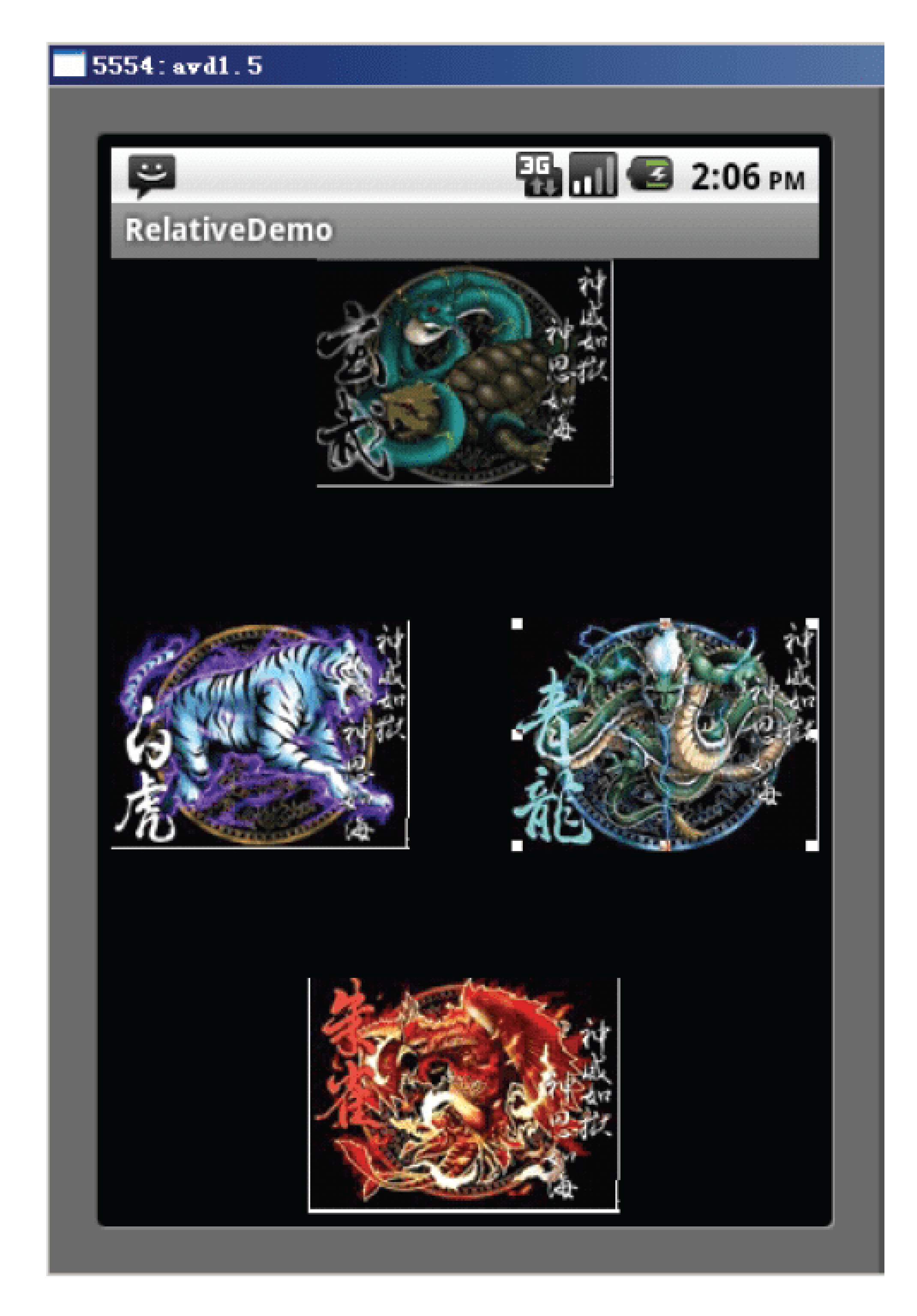


图 7.16　四神兽图

7.3　使用其他布局容器

除了使用 7.2 节中的 5 种布局容器类，我们还可以使用一些特殊的布局容器来进行屏幕的设计和布局，如使用 ListActivity、TabActivity 等，本节将讲解如何使用这些特殊的布局容器。

7.3.1　使用 TabActivity

拥有 Android 手机使用经验的读者对于图 7.17 肯定不陌生，这是联系人列表的显示方式，也许很多读者都很向往能够写出这么“酷”的布局来，本小节就讲解 Android 中标签页的使用。

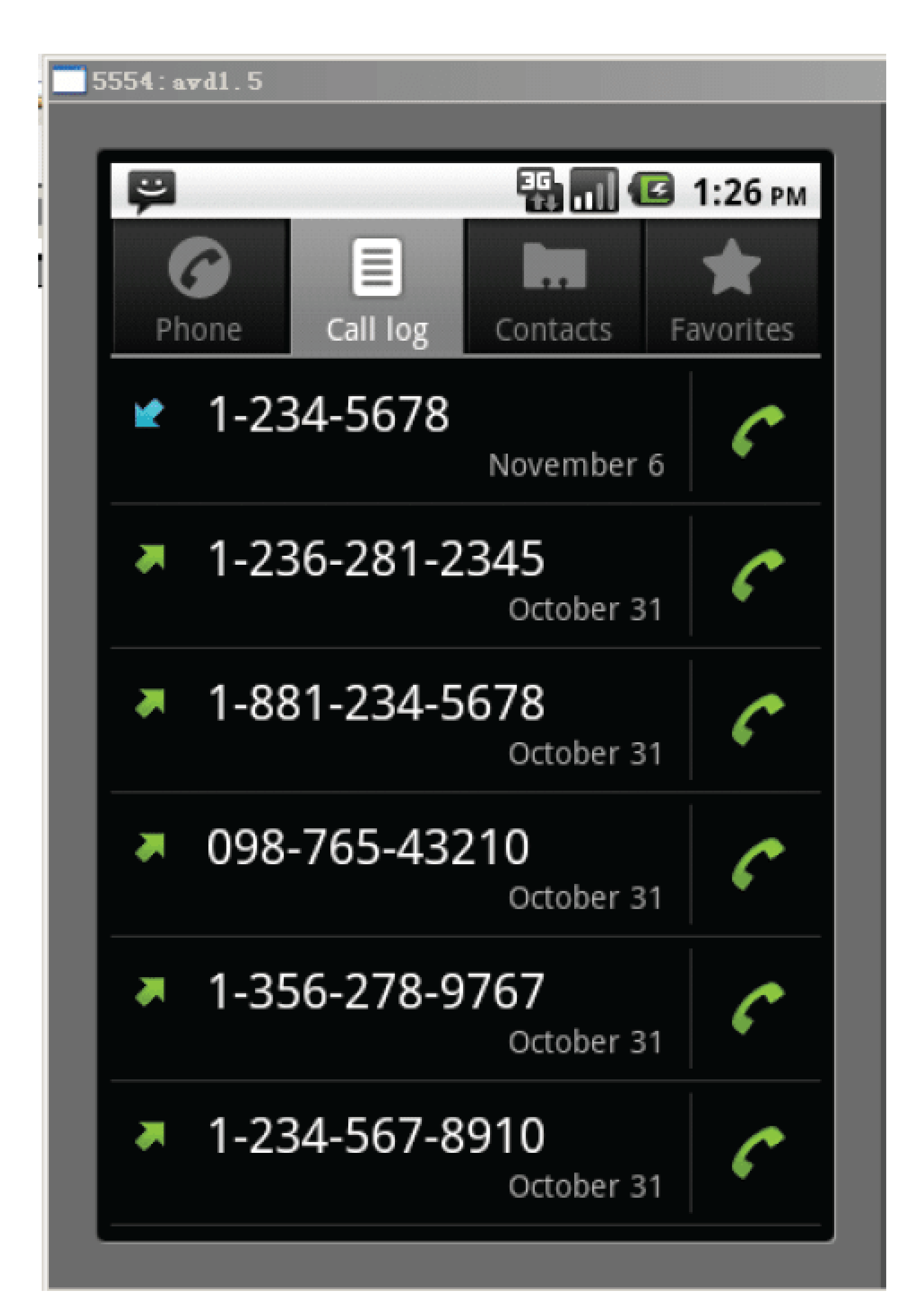


图 7.17　标签页的使用

标签页有两个主要的组成部分，一个是 TabActivity，另一个是 TabHost。TabHost 又由 TabSpec 组成，TabSpec 中包含了每个标签的标签名、显示内容等标签信息，是 TabHost 的嵌套类。

TabActivity 使用入门非常简单，但要使用好它却需要大家多花一些时间的。标签页中的每一个标签都是一个非常高效的视图容器，它可以由 xml 预先定义也可以由 TabFactory 产生。接下来我们开始学习使用 TabActivity 进行界面设计。

首先在布局文件中进行界面布局，一般我们使用 FrameLayout 进行标签页的布局，至于原因大家可以猜想一下，笔者在这里先“卖个关子”。让我们一起来观察如下布局

文件：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/tabcontent"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
    <!-- 布局 1  -->
    <LinearLayout
    android:id="@+id/tab1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="布局 1，来自线性布局 tab1"
    android:textSize="50sp"
    />
    </LinearLayout>
    <!-- 布局 2  -->
    <LinearLayout
    android:id="@+id/tab2"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    >
    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="布局 2，来自线性布局 tab2"
    android:textSize="30sp"
    />
    <AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
    </LinearLayout>

</FrameLayout>
```

在这个 FrameLayout 下包含有两个 LinearLayout，id 分别为 tab1 和 tab2。通过 7.2.3 小节的讲解相信读者朋友们对本布局文件会提出质疑：这样的显示效果不是 tab2 完全覆盖 tab1 嘛！

不要担心，TabHost 会帮你组织这些布局的。使用 TabActivity 需要如下几个步骤：

（1）继承 TabActivity。

（2）获得 TabHost 对象。

（3）实例化布局对象。

（4）创建并设置 TabSpec 对象。

（5）向 TabHost 中添加 TabSpec 完成标签页的使用。

接下来就进入代码实践过程吧！

1．继承TabActivity

这一步骤非常简单，只需将 extends Activity 改为 extends TabActivity 就可以了。同时

将 import android.app.Activity 改为 import android.app.TabActivity。

2．获得TabHost对象

在 TabActivity 父类中已经完成了 TabHost 的创建，我们只需使用：

```
This.getTabHost()
```

就可以得到它的使用对象了。

3．实例化布局对象

接下来将我们要显示的布局类实例化，这里可以细分为两小步：
（1）获得 LayoutInflater
通过使用 LayoutInflater.from(this)获得 inflater 对象：

```
LayoutInflater inflater = LayoutInflater.from(this);
```

（2）使用 LayoutInflater 实例化布局
实际上实例化布局在每一个 Activity 中都要进行，这些工作都包含在 setContentView()中，在使用 TabActivity 时不需要调用 setContentView()方法，所以我们需要自己来实例化布局。具体的方法为：

```
LayoutInflater.inflate(int resource, ViewGroup root)
```

这里有两个参数：
（1）int resource，资源的 id，如 R.id.xxx。
（2）视图容器类，ViewGroup 对象，这里为 TabHost.getTabContentView()。

4．创建并设置TabSpec对象

首先创建一个新的 TabSpec 对象：

```
TabHost.newTabSpec(String tag)
```

这里的参数是 TabSpec 的标签，在显示时没有什么作用。
接着，设置标签头，如果仅仅显示文字可以使用：

```
TabSpec.setIndicator(CharSequence label)
```

如果你希望显示文字和图片则使用：

```
TabSpec.setIndicator(CharSequence label, Drawable icon)
```

第一个参数是文字，第二个参数是图片。
最后设置需要显示的内容，如果要显示 xml 预定义的视图请使用：

```
TabSpec.setContent(int viewId)
```

如果使用 TabContentFactory 则使用：

```
TabSpec。setContent(TabContentFactory contentFactory)
```

关于 TabContentFactory 的使用会在之后进行讲解。

5. 向TabHost中添加TabSpec

添加 TabSpec 的工作类似于 setContentView()，只有向 TabHost 中添加了 TabSpec 才能正确显示出内容。方法如下：

```
TabHost.addTab(TabSpec tabSpec)
```

按照以上 5 个步骤，一个完整的 TabActivity 代码为：

```
import android.app.TabActivity;
import android.view.LayoutInflater;
import android.widget.TabHost;
import android.widget.TabHost.TabSpec;
import ......                              //省略部分导入

public class TabActivityDemo extends TabActivity
{
    TabHost m TabHost;
     @Override
    public void onCreate(Bundle savedInstanceState)
     {
        super.onCreate(savedInstanceState);
        m_TabHost = this.getTabHost();  //获得 TabHost 对象
        LayoutInflater inflater = LayoutInflater.from(this);   //获得布局
        inflater.inflate(R.layout.main, m TabHost.getTabContentView());
        TabSpec spec1 = m TabHost.newTabSpec("tab1").setIndicator("tab1").
        setContent(R.id.tab1);             //创建 TabSpec 对象
       TabSpec spec2 = m TabHost.newTabSpec("tab2").setIndicator("tab2",
       getResources().getDrawable(R.drawable.plus)).setContent(R.id.
       tab1);                              //标签中包含文字和图片
       TabSpec spec3 = m_TabHost.newTabSpec("tab3").setIndicator("tab3").
       setContent(R.id.tab2);
       TabSpec spec4 = m TabHost.newTabSpec("tab4").setIndicator("tab4",
       getResources().getDrawable(R.drawable.drawings)).setContent(R.id.
       tab2);

       m TabHost.addTab(spec1);            //向 TabHost 中添加 TabSpec 对象
       m TabHost.addTab(spec2);
       m_TabHost.addTab(spec3);
       m_TabHost.addTab(spec4);
     }
}
```

运行程序后显示如图 7.18 所示。





图 7.18　标签页显示效果

7.3.2　自定义 TabHost

直接使用 TabActivity 固然是方便，但是使用起来毕竟有很多限制，感觉不是很随心。那么可不可以自定义 TabHost 呢?答案是完全可以！

在自定义 TabHost 时需要注意，创建 TabHost 时需要以下 3 个步骤：

（1）在 xml 资源文件中创建 TabHost 节点，并将 id 设置为 tabhost。

（2）创建 TabWidget 子节点，并设置 id 为 tabs。

（3）创建 FrameLayout 子节点，用于显示内容，其 id 为 tabcontent。

学习了以上 3 个步骤后，我们开始编写 xml 代码，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@android:id/tabhost"   //注意 id 表述方式，一定要是"@android:id/
tabhost"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
    <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
        <TabWidget
        android:id="@android:id/tabs"
                            //注意 id 表述方式，一定要是"@android:id/tabs"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

        <FrameLayout
        android:id="@android:id/tabcontent"
```

```
                    //这里的 id 一定要是"@android:id/tabcontent"
        android:layout width="fill parent"
        android:layout height="fill parent">
            <!-- 布局 1  -->
            <LinearLayout
            android:id="@+id/tab1"
            android:layout width="fill parent"
            android:layout height="fill parent"
            >
            ......                  //自定义布局
            </LinearLayout>
            <!-- 布局 2  -->
            <LinearLayout
            android:id="@+id/tab2"
            android:layout width="fill parent"
            android:layout height="fill parent"
            >
            ......                  //自定义布局
            </LinearLayout>
        </FrameLayout>
    </LinearLayout>
</TabHost>
```

以上的布局文件中笔者已经做出注释，其中有 3 个节点的 id 属性不可修改，这是因为在 TabHost 初始化时需要使用这 3 个属性，如果被修改，则 SDK 无法识别该节点，在创建 TabHost 时会报错。

在代码中使用 TabHost 与 TabActivyt 比较相似，不同的只有开始的两个步骤，其具体步骤如下：

（1）使用 setContentView()方法显示界面。

（2）TabHost 对象获得并设置。

（3）创建并设置 TabSpec 对象。

（4）向 TabHost 中添加 TabSpec 完成标签页的使用。

与使用 TabActivity 相比较不难发现，自定义 TabHost 时不需要继承 TabActivity 了，只需要简单继承 Activity 就可以了，这无疑给我们编程提供了更大的自由发挥的空间。因为我们知道继承虽然会给我们的编程带来很大程度的方便，但也同样带来了很多的条条框框的限制。

让我们着重来看第二步获得 TabHost 对象，这里的获得 TabHost 对象的方法与使用 TabActivity 时又不一样了，具体方法为：

```
m_TabHost = (TabHost)findViewById(android.R.id.tabhost);
```

我们发现得到 TabHost 的方法是最简单、最常见的 findViewById()方法！这里的参数就是 android.R.id.tabhost，也就是在 xml 资源文件中我们实现定义的：

```
android:id="@android:id/tabhost"
```

需要注意的是，在获得了 TabHost 之后，我们需要调用：

```
TabHost.setup()
```

调用了该方法之后，TabHost 才设置完成可以正常使用。而在使用 TabActvity 时则不

需这一步，因为在 getTabHost()方法中已经完成了设置的工作。

接下来的两个步骤与之前讲解的完全一样，这里就不再赘述了，让我们一起来看一下最后的代码：

```
import ......                                             //省略部分导入
import android.widget.TabHost;
import android.widget.TabWidget;
import android.widget.TabHost.TabSpec;

public class TabDemo extends Activity implements TabHost.TabContentFactory
{

     private TabHost m TabHost;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main1);

        m TabHost = (TabHost)findViewById(android.R.id.tabhost);
                                                          //获得 TabHost 对象
        m_TabHost.setup();                                //设置 TabHost

        TabSpec spec1 = m TabHost.newTabSpec("tab1").setIndicator("tab1").
        setContent(R.id.tab1);
        TabSpec spec2 = m TabHost.newTabSpec("tab2").setIndicator("tab2").
        setContent(R.id.tab1);
        TabSpec spec3 = m TabHost.newTabSpec("tab3").setIndicator("tab3").
        setContent(R.id.tab2);
        TabSpec spec4 = m TabHost.newTabSpec("tab4").setIndicator("tab4").
        setContent(R.id.tab2);
        //下面两个标签页是通过 TabContentFactory 产生的
TabSpec spec5 = m_TabHost.newTabSpec("tab5").setIndicator("tab5").
        setContent(this);
        TabSpec spec6 = m_TabHost.newTabSpec("tab6").setIndicator("tab6").
        setContent(this);

        m TabHost.addTab(spec1);
        m TabHost.addTab(spec2);
        m TabHost.addTab(spec3);
        m TabHost.addTab(spec4);
        m TabHost.addTab(spec5);
        m TabHost.addTab(spec6);
    }

     @Override
     public View createTabContent(String arg0)
     {
         TextView tv = new TextView(this);            //动态生成标签页内容
         tv.setText("我来自 TabContentFactory! 动态生成! ");
         tv.setTextSize(25);
         LinearLayout ll = new LinearLayout(this);
         ll.addView(tv);
         return ll;
     }
}
```

通过 TabContentFactory 产生的 TabContent 好处是动态生成，每个 Widget 都有其自己的 Id，也就是说每个 Widget 都是不一样的。这样的好处是相同的布局下，我们可以操作

不同的组件，比如我们常用的 QQ，每个聊天框布局都一样，但是每个聊天框中显示的内容都不同。如果使用 xml 预定义则无法很好地达到目的。

运行程序后，效果显示如图 7.19 所示。





图 7.19　自定义 TabHost

本小节的最后再教读者朋友们一个小技巧，我们发现一旦标签页多了以后，显示起来会显得非常密集，非常不美观，也不利于用户的操作，这个时候只需做一些小小的完善，我们的程序就更友好了。我们需要如下两个步骤：

（1）在 xml 资源文件中为 TabWidget 节点包裹一层 HorizontalScrollView，这样它就可以水平滚动了，如下所示：

```
<HorizontalScrollView
            android:layout width="fill parent"
            android:layout height="wrap content"
            android:fillViewport="true"
            android:scrollbars="none"
            >
                <TabWidget
                android:id="@android:id/tabs"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
            </HorizontalScrollView>
```

其他的 xml 代码无需做任何改变。

（2）在 Java 代码中添加一个设置，限定一个屏幕宽度只显示规定个数的标签。例如，规定只显示 4 个标签，这时我们可以编写如下方法：

```
public void setParam()
    {
        int count = 0;
        TabWidget tabWidget = m_TabHost.getTabWidget();//获得 TabWidget 对象
        count = tabWidget.getChildCount();             //获得标签个数
        DisplayMetrics dm = new DisplayMetrics();
```

```
        getWindowManager().getDefaultDisplay().getMetrics(dm);//获得屏幕参数

        int screenWidth = dm.widthPixels;          //获得屏幕宽度
        int screenHeight = dm.heightPixels;        //获得屏幕高度

        if (count > 0)
        {
            for (int i = 0; i < count; i++)
            {
                tabWidget.getChildTabViewAt(i).getLayoutParams().width=(
                        screenWidth / 4);          //设置宽度为屏幕宽的1/4
                tabWidget.getChildTabViewAt(i).getLayoutParams().
                height=( (screenHeight - 40) / 12);//设置高度为屏幕宽的1/12
            }
        }
    }
```

在 onCreate()方法的最后调用这个函数，此时设置生效，预期的目的就达成了，显示如图 7.20 所示。



图 7.20　制作可滚动的标签

7.3.3　使用对话框

在进行界面编程的时候我们经常要使用对话框与用户进行更友好的交互，很多使用者都以为它是一种 Widget，实际上它更类似于 Activity，只不过它只能以弹窗的形式显示。对话框的功能非常强大，你可以通过对话框的按钮来判断用户的操作，也可以在对话框中添加布局和 Widget，以便实现更多的功能。接下来我们就一起来学习这些神奇的对话框到

底是如何使用的？

Dialog 是所有对话框的基类，在平时我们真正使用的对话框有两种：AlertDialog 以及 ProgressDialog。

1. AlertDialog

使用对话框时不能通过其构造函数，也就是 new AlertDialog()方法来新建，因为该方法是 protect 类型，我们必须使用 AlerDialog.builder 来达到我们的目的。AlertDialog.builder 是 AlertDialog 的内部静态类，通过其一系列的 set 方法，我们可以完成一个 AlertDialog 的创建。

与 Activity 一样，Dialog 也是有生命周期的，我们可以主动调用的两个函数分别是：

- showDialog(int id)：使 Dialog 对话框出现。
- dismissDialog(int id)：使 Dialog 对话框消失。

在调用 showDialog()方法时，系统会回调 onCreateDialog(int id)方法。所以一般我们的 Dialog 新建工作就在 onCreateDialog()方法中完成。当然你也可以重写 onPrepareDialog()方法，因为在这个时候已经新建完毕但还没有真正显示，这个时候你可以对其进行修改，如修改其中的提示信息等。

那么接下来我们就开始真正的新建过程了。在新建 AlertDialog 时，我们一般需要设置以下几个重要部分：

（1）setTitle(String name)：设置标题，显示在对话框的 title 位置。

（2）setMessage(String message)：设置消息，这个是显示在对话框中的信息。

（3）setIcon(int resId)：设置图片，参数是资源的 ID。

（4）setPositiveButton(String name，OnClickListener listener)：设置“确定”按钮。

（5）setNegativeButton(String name，OnClickListener listener)：设置“取消”按钮。

（6）setNeutral Button(String name，OnClickListener listener)：设置“忽略”按钮。

最后不要忘记调用 create()方法使设置生效。当然在“确定”、“取消”、“忽略”3 个按钮中你可以选择设置其中的一项或几项。

接下来请阅读以下代码：

```
@Override
    protected Dialog onCreateDialog(int id)
    {
        switch (id)
        {
        case DIALOG 1:
            return new AlertDialog.Builder(DialogDemo.this)
                                    .setTitle("DIALOG_1")
                                    .setMessage("这是对话框 1")
                                    .setIcon(android.R.drawable.ic dialog
                                    info)
                                    .setPositiveButton("确定", new OnClickLis
                                    tener()
                                    {
                                        @Override
                                        public void onClick(DialogInterface
                                        arg0, int arg1)
                                        {
                                            //在确定时添加你希望进行的操作
                                        }
```

```
                                  })
                                  .setNegativeButton("取消", null)
                                  .create();
            default:
                return null;
            }
        }
```

这样就完成了一个最简单的 AlertDialog 的创建了。

在 onCreate()方法中主动调用 showDialog()方法就可以展示对话框了：

```
public static final int DIALOG_1 = 1;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        showDialog(DIALOG_1);
}
```

运行以上代码，我们会看到如图 7.21 所示的效果。



图 7.21　最简单的对话框

接下来让我们进一步丰富对话框中的内容，如使用弹出对话框弹出一个菜单以供用户选择。这时我们可以选择单选框或多选框，如果需要使用单选框，则语法格式如下所示：

```
public void showDialog2()
    {
        new AlertDialog.Builder(this)
        .setTitle("Dialog2-单选框")
        .setIcon(android.R.drawable.ic_dialog_info)
        .setSingleChoiceItems(new String[] { "Item1", "Item2" }, 0,null)
                                                          //设置选项
        .setNegativeButton("取消", null)
        .setPositiveButton("确定", new OnClickListener()
```

```
            {
                @Override
                public void onClick(DialogInterface arg0, int arg1)
                {
                    showDialog3();
                }
            })
        .show();
    }
```

这里的关键方法是：

```
AlertDialog.Builder.setSingleChoiceItems(CharSequence[] items, int chec
kedItem, OnClickListener listener)
```

在 listener 中你可以完成所有你希望进行的工作，这里就简单设置为 null。运行后效果如图 7.22 所示。



图 7.22　单选框

如果你希望提供用户多个选择，可以将：

```
AlertDialog.Builder.setSingleChoiceItems(CharSequence[] items, int chec
kedItem, OnClickListener listener)
```

改为：

```
AlertDialog.Builder.setMultiChoiceItems(CharSequence[] items, boolean[]
checkedItems, OnMultiChoiceClickListener listener)
```

修改后，得到如下代码段：

```
public void showDialog3()
    {
        new AlertDialog.Builder(this)
        .setTitle("Dialog3-复选框")
        .setMultiChoiceItems(new String[] { "Item1","Item2","Item3",
```

```
            "Item4" }, null, null)
            .setPositiveButton("确定", new OnClickListener()
            {
                @Override
                public void onClick(DialogInterface arg0, int arg1)
                {
                    showDialog4();
                }
            })
            .setNegativeButton("取消", null)
            .show();
        }
```

运行后得到如图 7.23 所示的多选框效果。



图 7.23　多选框效果图

到这里也许你觉得对话框的功能已经够强大了，但是如果你有自己的数据源需要显示呢？例如，使用对话框显示你的好友列表、选择删除好友等操作，难道你要一个个填写 String 数组吗？

这个时候我们可以通过自己定义的 ListView 来显示，并且自定义的 ListView 我们可以更好地进行界面的设计和响应，通过 setView()方法我们可以将自定义的视图显示到 Dialog 上：

```
public void showDialog4()
    {
        //listItem 用来存放菜单项
        ArrayList<HashMap<String, String>> listItem = new ArrayList<HashMap
        <String, String>>();
        HashMap<String, String> map1 = new HashMap<String, String>();
        map1.put("ItemName", "添加");
        HashMap<String, String> map2 = new HashMap<String, String>();
        map2.put("ItemName", "删除");
        HashMap<String, String> map3 = new HashMap<String, String>();
        map3.put("ItemName", "退出");
```

```
        listItem.add(map1);
        listItem.add(map2);
        listItem.add(map3);
        //数据源
        String[] from = new String[] {"ItemName" };
        //用作显示的 TextView
        int[] to = new int[] { android.R.id.text1 };
        //新建适配器
       SimpleAdapter adapter = new SimpleAdapter(
            this,
            listItem,
            android.R.layout.select_dialog_item,
            from,
            to);
       //新建 ListView
       ListView mListview = new ListView(this);
       mListview.setAdapter(adapter);
        new AlertDialog.Builder(this)
            .setTitle("DIALOG 4")
            .setMessage("Dialog4-自定义列表")
            .setIcon(android.R.drawable.ic_dialog_info)
            .setPositiveButton("确定", new OnClickListener()
            {
                @Override
                public void onClick(DialogInterface arg0, int arg1)
                {
                    showDialog5();
                }
            })
            .setNegativeButton("取消", null)
            .setView(mListview)
            .create().show();
    }
```

以上代码段显示了如何将一个自定义的 View 显示到 Dialog 中，甚至如果你需要，同样可以在 Dialog 中显示所有的 Widget 组合。运行以上代码我们可以得到如图 7.24 所示的效果。



图 7.24　自定义列表

你同样可以自定义多选框，如使用如下代码段：

```
public void showDialog5()
    {
        ......        //此处产生数据源，与 showDialog4()中相同
        String[] from = new String[] {"ItemName" };
        int[] to = new int[] { android.R.id.text1 };
       SimpleAdapter adapter = new SimpleAdapter(
           this,
           listItem,
           android.R.layout.select_dialog_multichoice,
           from,
           to);
       ListView mListview = new ListView(this);
       mListview.setAdapter(adapter);
       mListview.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
        new AlertDialog.Builder(this)
           .setTitle("DIALOG_5")
           .setMessage("Dialog5-自定义多选框")
           .setIcon(android.R.drawable.ic dialog info)
           .setPositiveButton("确定", new OnClickListener()
           {
               @Override
               public void onClick(DialogInterface arg0, int arg1)
               {
                   showDialog6();
               }
           })
           .setNegativeButton("取消", null)
           .setView(mListview)
           .create().show();
    }
```

注意其中的加粗部分，正是这两处修改使得我们刚才定义的单选列表改为了多选列表项，运行后显示效果如图 7.25 所示。



图 7.25　自定义多选框

2. 使用ProgressDialog

除了 AlertDialog 外，我们还经常使用 ProgressDialog 来为用户提供更好的操作体验，比如我们在使用地图服务时搜索地点一般需要很长的时间，而这个时候我们就可以使用进度对话框来提示用户“正在搜索，请稍后”等信息了。最简单的进度对话框如下所示：

```
public void showDialog6()
    {
            ProgressDialog dialog = new ProgressDialog(this);
            dialog.setMessage("Dialog6-进度对话框");
            dialog.setCancelable(true);
            dialog.show();
    }
```

运行后效果如图 7.26 所示。



图 7.26　进度对话框

7.3.4　使用滑动抽屉

在使用 Android 手机时，我们经常会操作滑动抽屉，如查看消息提示等。那么滑动抽屉是怎样实现的呢？实际上滑动抽屉是一种特殊的 Widget，不使用时它是隐藏的，需要时可以将之拖出进行操作。使用滑动抽屉时需要如下几个步骤。

1. xml部分

（1）在 xml 中使用<SlidingDrawer>节点，并设置属性。

（2）设置 handle 部分，即为手柄，可以理解为抽屉的把手。

（3）设置 content 部分，这是抽屉的内容，也就是将抽屉拉开时我们看到的界面部分。按照以上 3 个步骤，我们可以得到 xml 部分代码如下所示：

```
   <!-- 设置 SlidingDrawer -->
    <SlidingDrawer
        android:id="@+id/slidingdrawer"
        android:handle="@+id/handle"              //设置手柄的引用
        android:content="@+id/content"            //设置内容的引用
        android:layout width="fill parent"
        android:layout height="fill parent"
        android:orientation="vertical"            //设置纵向拉伸
    >
        <!-- 设置 handle -->
        <ImageButton
            android:id="@id/handle"
            android:background="@drawable/up"
            android:layout height="30dp"
            android:layout width="30dp"
        />
        <!-- 设置 content -->
        <LinearLayout
            android:orientation="vertical"
            android:id="@id/content"
            android:layout_width="fill_parent"
            android:layout height="fill parent"
            android:background="@drawable/bg1"
        >
            <TextView
                android:id="@+id/tv"
                android:text="这是一个滑动抽屉"
                android:textSize="50sp"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
            />
<Button
                android:id="@+id/btn"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="退出"
            />
        </LinearLayout>
    </SlidingDrawer>
```

如上所示，handle 和 content 都是可以自己定义的 Widget 或布局，只需在 Sliding Drawer 的：

```
android:handle="@+id/handle"
android:content="@+id/content"
```

属性中指向其 ID 就可以了。到这里 xml 部分的代码就完成了。

2. Java部分

在 Java 部分，你可以不写任何代码，但这样 SlidingDrawer 的功效无法得到更大程度上的体现。

在 SlidingDrawer 中你可以通过 findViewById()方法得到 SlidingDrawer 的操作对象，然

后通过以下 3 个方法进行事件的监听：

（1）监听抽屉被拉开事件：

```
SlidingDrawer.setOnDrawerOpenListener(OnDrawerOpenListener onDrawerOpen
Listener)
```

（2）监听抽屉被关闭事件：

```
SlidingDrawer.setOnDrawerCloseListener(OnDrawerCloseListener onDrawer
CloseListener)
```

（3）监听抽屉滑动中事件：

```
SlidingDrawer.setOnDrawerScrollListener(OnDrawerScrollListener
onDrawerScrollListener)
```

例如，我希望改变手柄拉开时和关闭时的方向，其代码段如下：

```
SlidingDrawer drawer;
ImageButton   btn;

       drawer = (SlidingDrawer) findViewById(R.id.slidingdrawer);
       btn    = (ImageButton)    findViewById(R.id.handle);

       drawer.setOnDrawerOpenListener(new OnDrawerOpenListener()
       {
            @Override
            public void onDrawerOpened()
            {
                btn.setBackgroundResource(R.drawable.down);
                btn.setAlpha(0);
            }
       });

       drawer.setOnDrawerCloseListener(new OnDrawerCloseListener()
       {
            @Override
            public void onDrawerClosed()
            {
                btn.setBackgroundResource(R.drawable.up);
                btn.setAlpha(0);
            }
       });
```

以上代码的作用是在 SlidingDrawer 拉开时，将手柄图片设为向下的箭头，在关闭时设为向上的箭头。

运行程序，未打开的抽屉如图 7.27 所示，打开着的抽屉如图 7.28 所示。

如果你不想上下拉动，而是希望左右拉动，完全可以！只需将 SlidingDrawer 的属性修改为：

```
<SlidingDrawer
          android:id="@+id/slidingdrawer"
          android:handle="@+id/handle"
          android:content="@+id/content"
          android:layout width="fill parent"
          android:layout height="fill parent"
          android:orientation="horizontal"
    >
```





图 7.27　关闭着的抽屉



图 7.28　打开的抽屉

注意代码中加粗部分的代码，将此属性修改后就可以将抽屉横向拉动了。当然，在 Java 代码中我们还需要改变一下使用的背景图片，将箭头从上下指向改为左右指向。运行后效果如图 7.29 所示。





图 7.29　横向拉动的抽屉

7.4 小　　结

本章讲解了 Android 中布局的使用。通过本章的学习，读者朋友们可以将之前学习的资源和组件进行一个有机的结合，更好地进行界面的设计工作。本章的难点是一些特殊的布局容器的使用，如 TabActivity、AlertDialog 以及 SlidingDrawer 的使用，这也是本章的重点内容。

下一章我们将进入第 3 篇。在第 3 篇中会讲解 Android 中常用的一些 API，如操作数据库、多媒体以及地图服务等。

第 3 篇　功能实现

第 8 章　Android 应用程序组成

本章是第 3 篇的开始，通过本章的学习，我们可以结构性地了解 Android 应用程序的重要组成部分，包括活动（Activity）、广播接收器（Broadcast Receiver）、服务（Service）和内容提供者（ContentProvider）。一个 Android 应用程序必定包含至少一个 Activity，其他的 3 个组成部分为可选部分。

8.1　深入理解 Activity

在之前的几个章节中，我们已经初步地认识并了解了 Android 应用中最重要的部分——活动（Activity）。每个 Activity 包含一个或者几个页面，在第 2 篇的学习中，我们知道 Activity 一般作为视图的容器而存在。我们知道一个优秀的应用经常由多个 Activity 组成，那么这些 Activity 是如何连接起来的呢？或者说我们是怎样从一个页面跳转到另一个页面的呢？接下来我们将学习 Activity 连接的纽带——意图（Intent）。

8.1.1　使用 Intent 连接 Activity

意图（Intent）被用来连接各个 Activity，也被用来在各个 Activity 中传递数据。在本小节中将学习：

（1）创建 Intent。

（2）使用 startActivity()调用 Intent 完成跳转。

（3）使用 startActivityForResult()方法调用 Intent。

（4）使用 Intent 在 Activity 中传递数据。

使用 Intent 完成 Activity 的跳转只需两个步骤。

1．创建Intent

在创建 Intent 时，我们可以使用如下构造方法：

```
Intent.Intent(Context packageContext, Class<?> cls)
```

也可以先构造一个未指向的 Intent，然后通过如下方法指定跳转的 Activity：

```
Intent.setClass(Context packageContext, Class<?> cls)
```

2．调用Intent

创建完成后，我们可以使用 startActivity()方法调用 Intent 以完成跳转，语法格式如下：

```
Activity.startActivity(Intent intent)
```

如果希望下一个 Activity 返回结果至本 Activity，则调用 startActivityForResult()方法，语法格式如下：

```
Activity.startActivityForResult(Intent intent, int requestCode)
```

接下来让我们通过一个实例加深对 Intent 的理解：

（1）首先创建一个工程，在工程中创建一个主 Activity。

通过之前章节的学习和实践，这一步相信读者朋友们独立完成没有什么问题。

（2）修改 Activity 的布局文件，增加一个按钮，按钮显示“跳转到 Activity1”。

（3）创建一个新的 Activity。在之前的学习中我们经常在创建工程的同时完成了 Activity 的创建，并没有独立创建 Activity 的经验，这里就稍稍讲解一下如何创建一个新的 Activity：

① 选中希望创建类的位置。例如，要在 com.wes.intentdemo 包中创建一个新的类，就先选中该包名并单击右键菜单中的 new|class 命令，如图 8.1 所示。

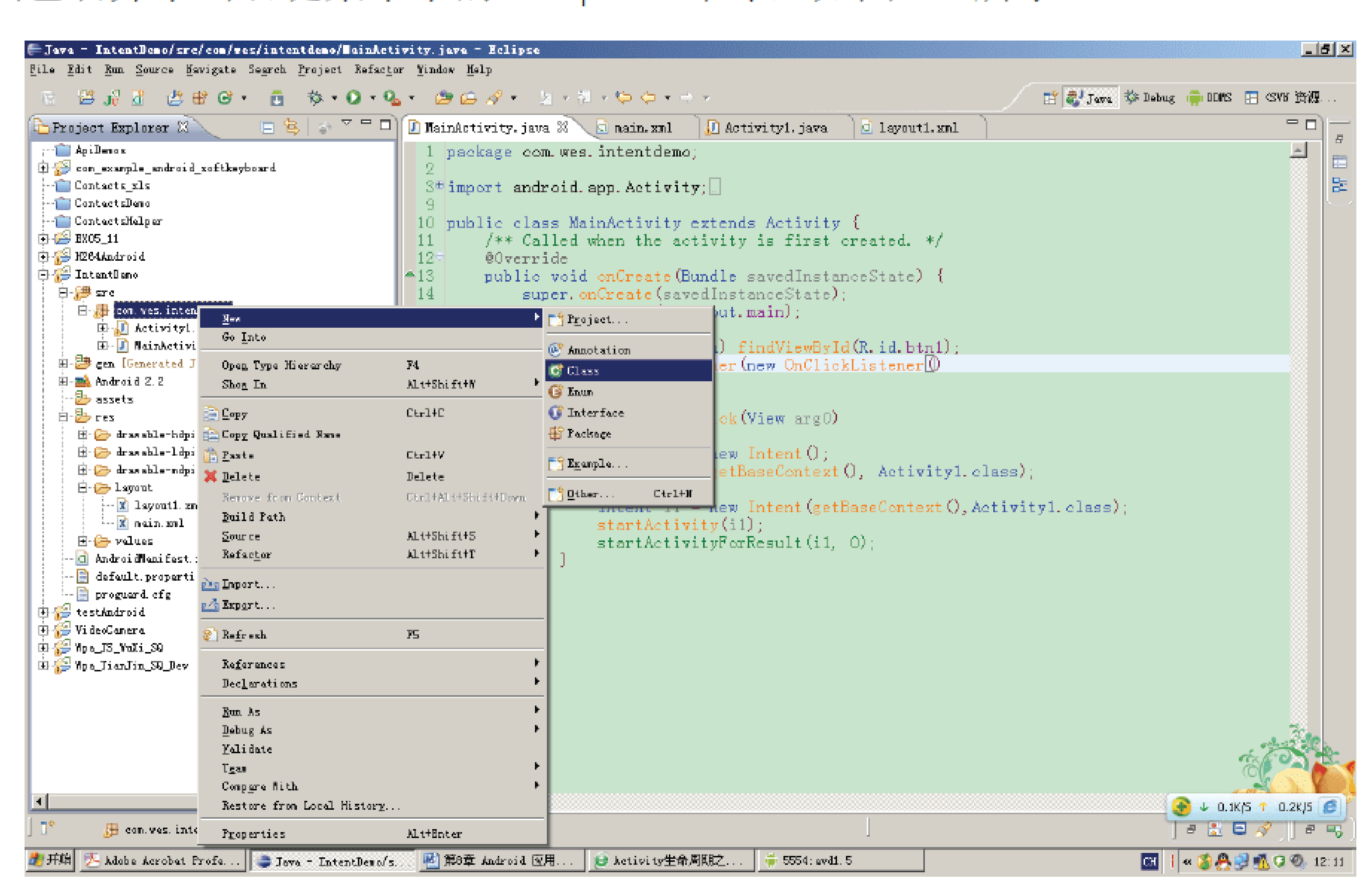


图 8.1　新建 Class

② 在弹出的 New Java Class 对话框中填入相关信息，最重要的就是类名，也就是 Name 输入框，如图 8.2 所示。

③ 在 Superclass 框中输入 android.app.Activity，这样就指定了该类继承自 Activity。

④ 单击 Finish 按钮，完成创建。

此时我们可以看到 Eclipse 为我们新建了一个名为 Activity1 的类，打开该类，我们发现只有如下 4 行代码：

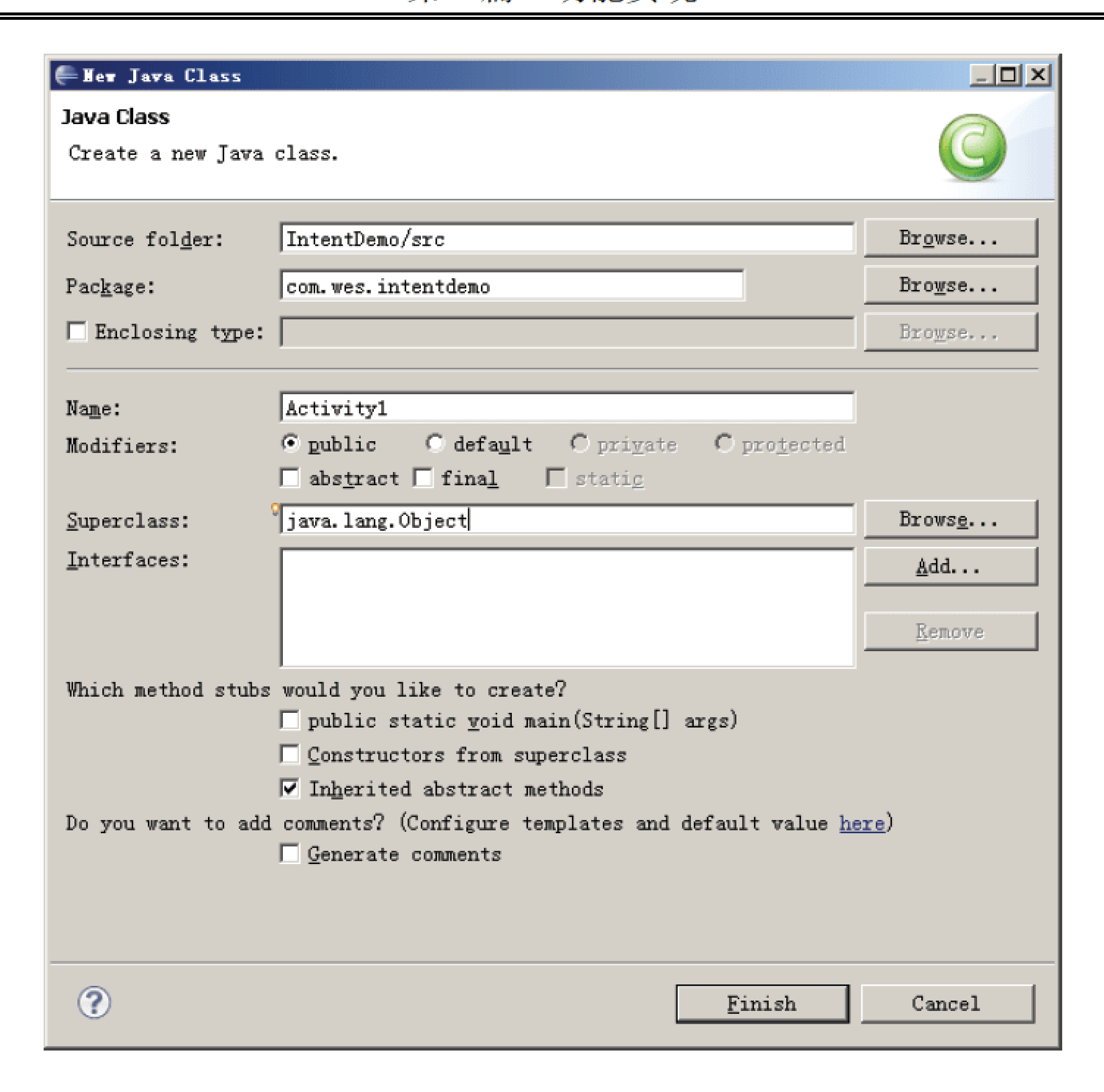


图 8.2　New Java Class 对话框

```
package com.wes.intentdemo;

import android.app.Activity;

public class Activity1 extends Activity
{
}
```

⑤ 在类中单击右键，在菜单中选择 Source 命令，如图 8.3 所示。

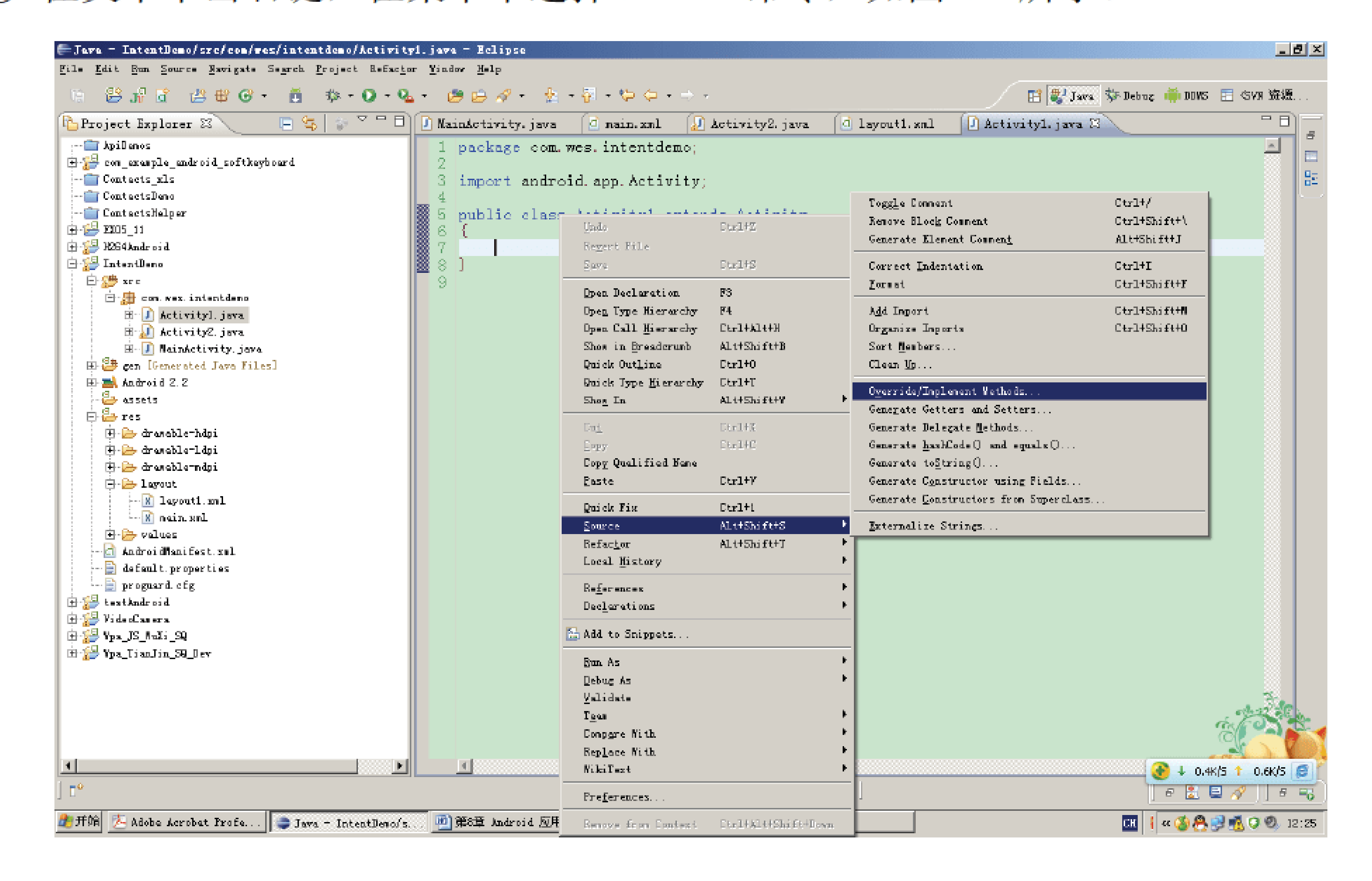


图 8.3　重写 onCreate()方法

在弹出的对话框中选择 onCreate(Bundle)方法，单击 OK 按钮，如图 8.4 所示。

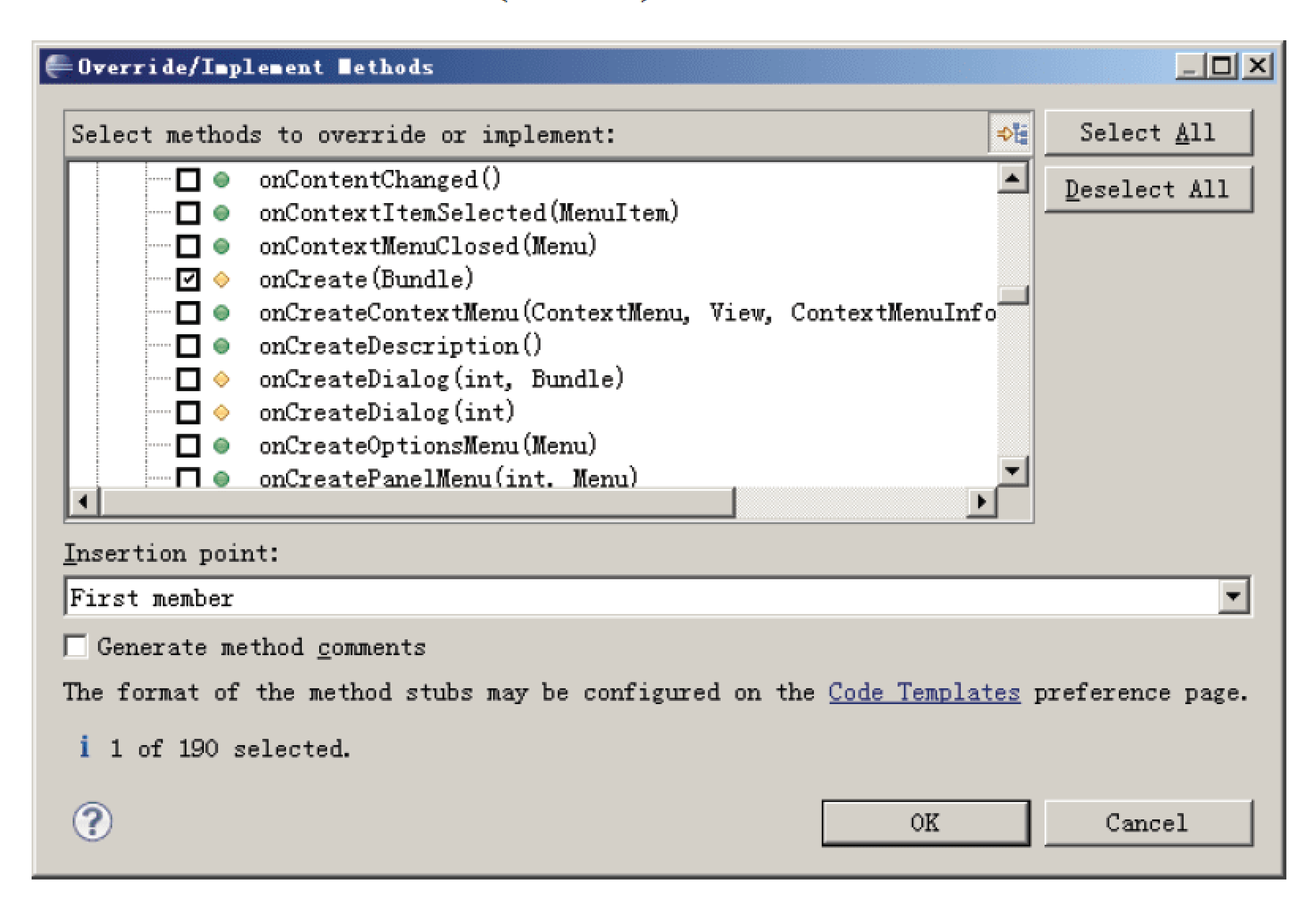


图 8.4　重写方法选择

这样我们熟悉的代码段又回来了！如下所示：

```
package com.wes.intentdemo;

import android.app.Activity;
import android.os.Bundle;

public class Activity1 extends Activity
{
     @Override
     protected void onCreate(Bundle savedInstanceState)
     {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
    }

}
```

接下来的工作大家应该都可以轻车熟路地完成了。第一步新建一个布局文件，布局中只需一个 TextView 显示“这是 Activity1”。接着在 onCreate()方法中添加：

```
setContentView (int resId)
```

这样一个新的 Activity 就创建完成了。

3. 添加Intent相关代码

在 MainActivity 的 onCreate()方法中添加如下代码：

```
Button btn1 = (Button) findViewById(R.id.btn1);
       btn1.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
```

```
            {
                Intent i0 = new Intent();  //新建一个 Intent
                i0.setClass(getBaseContext(), Activity1.class);
                                           //设置 Intent 跳转起始 Activity
                startActivity(i0);         //开始跳转
            }
        });
```

到这里，也许有的读者朋友们觉得现在本实例已经编写完成了，但实际上，如果你运行程序，会在 Log 中发现如下日志，如图 8.5 所示。

tag	Message
AndroidRuntime	FATAL EXCEPTION: main
AndroidRuntime	android.content.ActivityNotFoundException: Unable to find explicit activity class {com.wes.intentdemo/com.ve
AndroidRuntime	at android.app.Instrumentation.checkStartActivityResult(Instrumentation.java:1404)
AndroidRuntime	at android.app.Instrumentation.execStartActivity(Instrumentation.java:1378)
AndroidRuntime	at android.app.Activity.startActivityForResult(Activity.java:2817)
AndroidRuntime	at android.app.Activity.startActivity(Activity.java:2923)
AndroidRuntime	at com.wes.intentdemo.MainActivity$1.onClick(MainActivity.java:25)
AndroidRuntime	at android.view.View.performClick(View.java:2408)
AndroidRuntime	at android.view.View$PerformClick.run(View.java:8816)
AndroidRuntime	at android.os.Handler.handleCallback(Handler.java:587)
AndroidRuntime	at android.os.Handler.dispatchMessage(Handler.java:92)
AndroidRuntime	at android.os.Looper.loop(Looper.java:123)

图 8.5　ActivityNotFound 异常

该日志说明了一个问题，也就是 Android 系统无法发现要跳转的 Activity，问题出在哪里呢？因为我们没有在 AndroidManifest 注册文件中找到名为 Activity1 的 Activity，在前文我们提到：应用中的所有 Activity 都需要在注册文件中进行注册，否则会出现上述异常。解决该异常的办法就是在 AndroidManifest 文件中添加如下代码：

```
<activity android:name="Activity1"></activity>
```

添加后的注册文件如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.wes.intentdemo"
      android:versionCode="1"
      android:versionName="1.0">
   <uses-sdk android:minSdkVersion="8" />

   <application android:icon="@drawable/icon" android:label="@string/
   app_name">
      <activity android:name="MainActivity"
               android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
       <activity android:name="Activity1"></activity>
   </application>
</manifest>
```

注意<activity>节点的注册位置，必须在<application>节点之间，否则注册无效。此时再运行程序就可以成功了，如图 8.6 所示。





图 8.6　程序运行效果图

完成了 Intent 最基本的应用，让我们接着挖掘 Intent 的使用，在一个 Activity 跳转到另一个 Activity 时可以通过 Intent 传递数据，这也是 Intent 十分实用的一个功能。使用步骤分为两步：

（1）在起始 Activity 中存入需要传递的数据。语法格式为：

```
Intent.putExtra(String name, String value)
```

这里的两个参数 name 和 value 想必大家都不会陌生，它们以键值对的形式出现，类似于 HashMap，不过这里它们的类型是 Bundle 类型。Bundle 类型实际上是一种 HashMap 的再封装，是为了 Activity 而专门设计的。

（2）在目标 Activity 中取出 Intent 中携带的数据，语法格式为：

```
Intent.getExtras()
```

通过该方法可以得到一个 Bundle 对象，该对象中就包含有 Intent 携带的数据了：

```
Bundle.getString(String key)
```

接着再使用上述的 getString()方法，可以通过 key 参数得到该 key 所对应的值，而这个值就是我们最终需要的数据了。

接着让我们看一个实际的小例子，首先设计主 Activity 的布局，我们在布局文件中添加组件如表 8-1 所示。

表 8-1　主Activity布局组件列表

类　　型	ID	含　　义
TextView	Name	提示输入姓名
TextView	Age	提示输入年龄
TextView	Sex	提示输入性别
EditText	Name_in	姓名输入框
EditText	Age_in	年龄输入框
EditText	Sex_in	性别输入框
Button	Btn1	跳转按钮

最后我们使用 TableLayout 来更好地组织这些 Widget，想必大家可以独立完成了。

接着，在主 Activity 中添加如下代码：

```
EditText name_in = (EditText)findViewById(R.id.name);
         EditText age_in = (EditText)findViewById(R.id.age);
         EditText sex_in = (EditText)findViewById(R.id.sex);

         final String name = name_in.getText().toString();
         final String age = age_in.getText().toString();
         final String sex = sex_in.getText().toString();
```

```
        Button btn1 = (Button) findViewById(R.id.btn1);
        btn1.setOnClickListener(new OnClickListener()
         {
              @Override
              public void onClick(View arg0)
              {
                  Intent i = new Intent(getBaseContext(),Activity1.class);
                  i.putExtra("name",name);
                  i.putExtra("age",age);
                  i.putExtra("sex",sex);
                  startActivity(i);
              }
         });
```

关于如何取得数据这里就不再赘述，我们重点关注 onClick()事件中的操作。首先我们创建了一个从本 Activity 到 Activity1 的 Intent，接着向该 Intent 中加入数据，添加时需要为每个 Value 对应好一个 key。最后通过 startActivity(Intent)方法调用 Intent 以开始跳转。

接着我们要做的事情就是在 Activity1 中取得我们传递的数据：

（1）通过 getIntent()方法得到 Intent 对象。

（2）通过 getExtras()方法得到 Bundle 对象。

（3）通过 getString(String key)方法得到具体的数据。

按照以上步骤，代码段如下所示：

```
TextView show = (TextView) findViewById(R.id.show);

      Intent i = getIntent();                          //得到 Intent 对象
      Bundle bundle = i.getExtras();                   //得到 Bundle 对象
      String name = bundle.getString("name");          //得到 name 值
      String sex = bundle.getString("sex");            //得到 sex 值
      String age = bundle.getString("age");            //得到 age 值

      show.setText("你的信息如下:\n"+"姓名 :"+name+"\n 性别 :"+sex+"\n 年龄 :
      "+age);
```

最后运行程序，效果如图 8.7 所示。





图 8.7　使用 Intent 携带数据

在完成应用时，我们经常遇到这样的需求：我们希望从一个界面跳转到另一个界面进行相关的工作，如注册等，在注册页面完成输入相关信息后再返回到起始页面，并显示用户在注册页面输入的相关信息。难道我们需要使用一个 startActivity(Intent)跳转到一个 Activity，然后在目标 Activity 中重新新建一个 Intent，再次使用 startActivity(Intent)方法启动之前的 Activity 吗？这样似乎太麻烦了吧！这时我们可以使用另一种启动 Activity 的方式：

```
startActivityForResult（Intent intent）
```

从方法名我们也可以看出，这个方法是启动一个 Activity 并等待结果。这个方法在实际开发中经常被使用，使用它的主要步骤如下：

（1）新建 Intent，并使用 startActivityForResult()方法调用该 Intent。

（2）重写 onActivityResult()方法，在方法中处理返回结果。

（3）在目标 Activity 中，新建一个空指向的 Intent，并绑定数据。

（4）使用 setResult()方法，将 Intent 传递到结果中。

（5）调用 finish()方法结束目标 Activity。

在理解了以上 5 个步骤的基础上，我们再进行实际的代码部分讲解：

（1）新建 Intent 相信现在大家已经轻车熟路了，我们直接看 startActivityForResult()方法，其语法格式如下：

```
Activity.startActivityForResult(Intent intent, int requestCode)
```

这里有两个参数，第一个众所周知是 Intent 对象，第二个参数是请求码，用来标识这次请求，在第二步需要使用它。

（2）在 onActivityResult()方法中我们需要对返回的结果进行处理，该方法代码如下：

```
@Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
    data)
    {
        super.onActivityResult(requestCode, resultCode, data);
}
```

这里有 3 个参数：

第一个是请求码，用以标示本次结果对应哪个请求。

第二个参数是结果码，标准的结果码有两类：

RESULT_OK ：本次操作成功则返回该值。

RESULT_CANCELED：本次操作取消则返回该值。

如果你需要自己定义结果码那也是完全可行的，但是请不要与系统定义的结果码冲突，系统定义的 3 个结果码值分别为：

```
Public class android.app.Activity extends......{
Public static final int RESULT_CANCELED = 0;
Public static final int RESULT_OK = -1;
Public static final int RESULT_FIRST_USER = 1;
}
```

使用结果码可以区分操作成功与否，也可以区分该结果到底由哪个 Activity 返回。

第三个参数是 Intent data，该参数就是携带的返回结果，通过 Intent.getExtras()就可以

得到其中的数据了。

（3）在目标 Activity 中新建一个 Intent 对象，无需指定要跳转的 Activity，使用 Intent.putExtras()方法将数据与 Intent 绑定。

（4）将 Intent 传递到结果中去，使用方法：

```
Activity.setResult(int resultCode, Intent data)
```

这里有两个参数，第一个就是结果码，其意义前文已经讲解，第二个是 Intent 对象，其作用是保存数据。

（5）调用 Activity.finish()方法，结束本 Activity，这样 Android 系统就会调用之前重写的 onActivityResult()方法了。

我们接着通过一个小例子加深对 startActivityForResult()方法的理解。我们的需求很简单，从主 Activity 跳转到注册 Activity，在注册 Activity 中单击“确定”按钮返回到主 Activity，并将注册页面的信息显示出来。

首先，我们创建一个主 Activity，在主 Activity 中布局一个 TextView 用以显示信息，以及一个 Button，用以跳转。

接着新建一个注册 Activity，在注册 Activity 中增加一些 EditText 用以输入信息，其布局与上一个小例子类似，这里就不再赘述。

接着在主 Activity 中添加代码段如下所示：

```
Import ......                                          //导入略
public class MainActivity extends Activity {
    static final int REQUEST_CODE = 0 ;                //预定义请求码
    TextView show;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout1);

        show = (TextView) findViewById(R.id.show);     //获得 TextView 对象

        Button btn1 = (Button) findViewById(R.id.btn0);
        btn1.setOnClickListener(new OnClickListener()
         {
              @Override
              public void onClick(View arg0)
              {
                 Intent i = new Intent(getBaseContext(),Activity1.class);
                                                          //创建 Intent
                 startActivityForResult(i, REQUEST_CODE); //开始跳转
             }
        });
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
    data)
    {
        if (requestCode == REQUEST_CODE)               //判断请求码是否正确
        {
            if (resultCode == RESULT_OK)               //判断结果码是否正常
            {
```

```
                Bundle bundle = data.getExtras();//取得保存数据的 Bundle 对象
                String name = bundle.getString("name");
                String age  = bundle.getString("age");
                String sex  = bundle.getString("sex");
                show.setText("您的信息如下: \n"+"姓名 : "+name+"\n 性别 :
                "+sex+"\n 年龄 : "+age);
            }
        }
        super.onActivityResult(requestCode, resultCode, data);
    }
```

注册 Activity 的代码段如下所示：

```
@Override
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        EditText name_in = (EditText)findViewById(R.id.name);
        ......                                   //获得其他 EditText 对象

        final String name = name in.getText().toString();
        ......                                   //获得其他输入框中的信息

        Button btn = (Button)findViewById(R.id.btn1);
        btn.setOnClickListener( new OnClickListener()
         {
             @Override
             public void onClick(View arg0)
             {
                  Intent i = new Intent();       //新建 Intent
                  Bundle bundle = new Bundle(); //新建 Bundle 对象用以保存数据
                  bundle.putString("name",name);
                  bundle.putString("age",age);
                  bundle.putString("sex",sex);  //将数据保存到 Bundle 中
                  i.putExtras(bundle);           //将 Bundle 与 Intent 绑定
                  setResult(RESULT_OK,i);        //将 Intent 设置到结果中
                  finish();                      //结束本 Activity
             }
        });
```

运行以上代码，得到的效果如图 8.8、图 8.9、图 8.10 所示。



图 8.8　登录时显示



图 8.9　注册时的界面



图 8.10 单击“确定”按钮后携带结果返回

8.1.2 Activity 的生命周期

Activity 在前面的学习中我们经常使用，但是读者朋友们是否真的了解 Activity 呢？到目前为止我们还只是使用了 Activity 的诸多回调方法中的一个 onCreate()方法而已，诸如 onStart()、onPause()、onStop()等方法都还没有使用。

本小节将详细介绍这些 Activity 的生命周期。

首先，Activity 包括如下生命周期：

（1）onCreate()

（2）onStart()

（3）onResume()

（4）onPause()

（5）onStop()

（6）onDestroy()

（7）onRestart()

英文水平比较好的读者也许能直接从字面上得到这些方法的意义了。在 Activity 的 7 个生命周期中让我们先撇开 onRestart()方法，因为它比较特殊。剩下的 6 个生命周期我们可以一对一对地来理解。

1. onCreate()和onDestroy()

一个完整的 Activity 生命周期是由系统调用 onCreate()开始，并由调用 onDestroy()结束。Activity 在 onCreate()中设置所有“全局”状态以完成初始化，所以我们经常在该方法中完成界面的初始化工作。在 onDestroy()中 Activity 释放所有的系统资源，所以在该方法中我们经常会调用一些 close()或者 release()方法以释放资源，一旦进入 onDestroy 方法，该 Activity 的所有信息被销毁，将无法被还原。

2. onStart()和onStop()

onStart()方法和 onStop()方法决定了这个 Activity 是否可见。一旦系统调用了 onStart()方法，该 Activty 就对用户可见了，但是要注意的是，这里的可见并不意味着对用户是可操作的，再次强调：在 onStart()方法调用后仅仅意味着 Activity 可见！而 onStop()方法正好相反，一旦系统调用了 onStop()方法，那么该 Activity 就对用户不可见了。虽然此时不可

见，但它仍然保留所有的状态和成员信息。当然，如果其他地方需要内存，系统经常会杀死这个 Activity。

3. onResume()和onPause()

onResume()方法和 onPause()方法决定了这个 Activity 是否可操作。从系统调用 onResume()方法开始，用户就可以操作该 Activity 了，此时用户可以与 Activity 进行各种交互动作。一旦系统调用了 onPause()，Activity 就进入了不可操作状态，当然此时 Activity 仍然是可见的。

也许用语言来描述这个情况并不是很好理解，那么我们就举一个小例子：在你操作一个 Activity 时经常会遇到 AlertDialog 弹出，在弹出 AlertDialog 时，我们只能操作该对话框，而不能操作原来的 Activity，这时就是 Activity 可见而不可操作的状态了。这时系统就会调用 onPause()方法了，也就是说，Activity 经常会在 onResume()和 onPause()之间切换，理解了这点对于以后的编程很有帮助。

4. onRestart()

当系统调用了 onStop()方法以后，Activity 不可见了，但此时 Activity 的各种资源和信息仍然存在，所以系统不需要再次执行 onCreate()方法创建资源，此时可以通过 onRestart()方法重新开始该 Activity。

如果你还是无法完全理解上述的生命周期，没有关系，让我们通过图 8.11 来更直观地认识它。

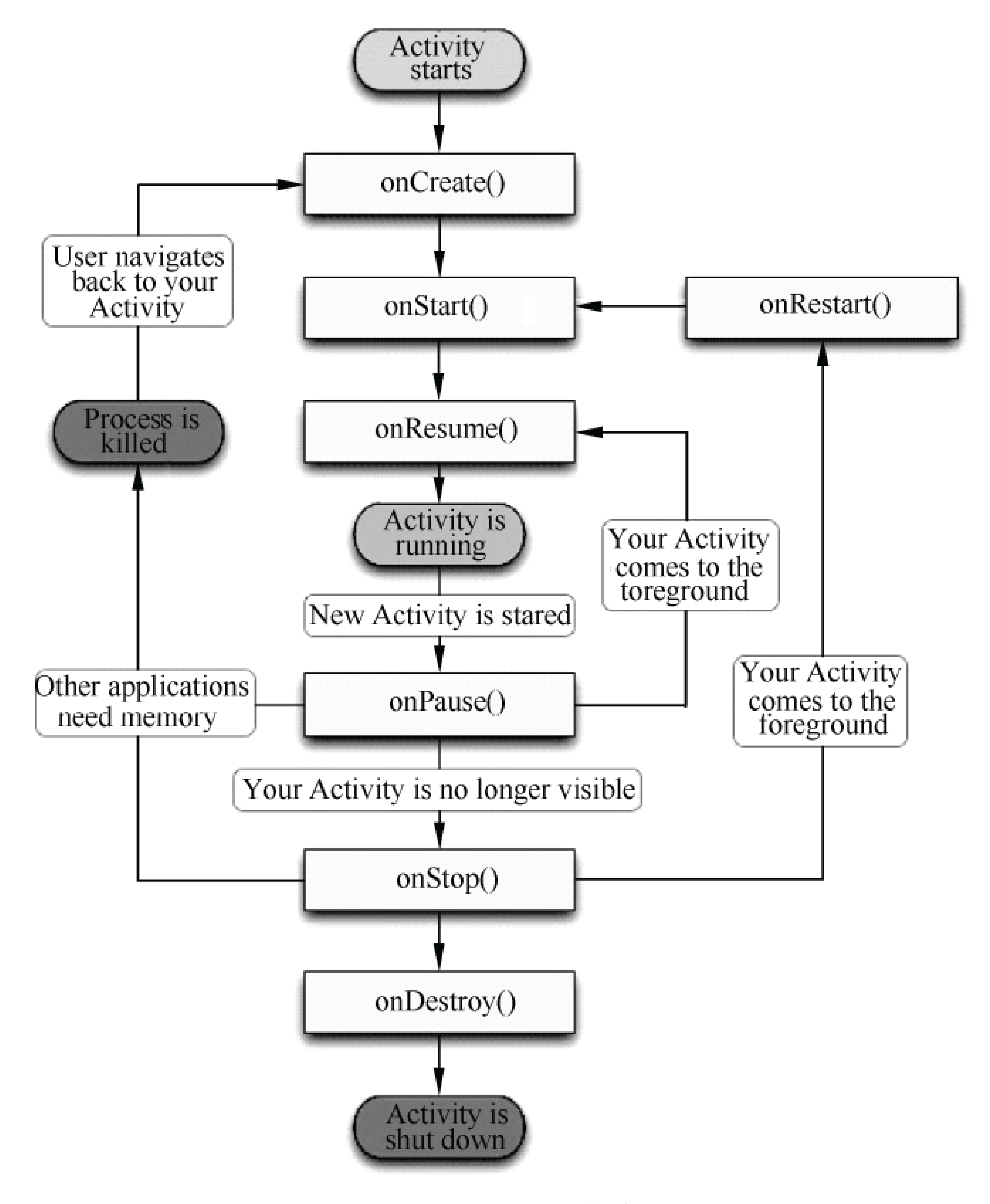


图 8.11　Activity 生命周期

通过图 8.11 可以再次直观地理解 Activity 生命周期了，接下来我们用一个实例来验证 Android 是怎样管理 Activity 的生命周期的。

让我们新建两个 Activity1 分别命名为 ActivityA 和 ActivityB，并重写这两个 Activity 的 7 个生命周期，在执行时添加日志打印。

1．ActivityA代码

```
public class ActivityA extends Activity {

     String TAG = "LifeCycle";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView tv = (TextView) findViewById(R.id.text);
        tv.setText("这里是 ActivityA! ");
        Button btn = (Button)findViewById(R.id.button);
        btn.setOnClickListener(new OnClickListener()
         {
             @Override
             public void onClick(View arg0)
             {
                 Intent i = new Intent(ActivityA.this,ActivityB.class);
                 startActivity(i);
             }
         });
        Log.d(TAG," A----------->onCreate");
    }

    public void onStart()
    {
        super.onStart();
        Log.d(TAG," A----------->onStart");
    }

    public void onRestart()
    {
     super.onStart();
     Log.d(TAG," A----------->onRestart");
    }

    public void onResume()
    {
     super.onResume();
     Log.d(TAG," A----------->onResume");
    }

    public void onPause()
    {
        super.onPause();
        Log.d(TAG," A----------->onPause");
    }

    public void onStop()
    {
        super.onStop();
        Log.d(TAG," A----------->onStop");
```

```
    }

    public void onDestroy()
    {
        super.onDestroy();
        Log.d(TAG," A----------->onDestroy");
    }
}
```

ActivityA 中 Button 的单击事件是从 ActivityA 跳转到 ActivityB，这样 ActivityA 必定会经历一部分的生命周期，这样我们在日志上就可以看出端倪了。

2. ActivityB代码

```
public class ActivityB extends Activity {
    String TAG = "LifeCycle";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        Log.d(TAG," B----------->onCreat");
        LinearLayout ll = new LinearLayout(this);
        Button btn = new Button(this);
        final TextView  txt = new TextView(this);
        ll.addView(txt);
        ll.addView(btn);
        setContentView(ll);
        this.setTitle("ActivtyB");
        txt.setText("这里是 ActivityB! ");
        btn.setText("跳转到 ActivityA");
        btn.setOnClickListener(new OnClickListener()
        {
         @Override
         public void onClick(View arg0)
         {
              Intent i = new Intent(ActivityB.this,ActivityA.class);
              startActivity(i);
         }
        });
    }

    public void onStart()
    {
       super.onStart();
       Log.d(TAG," B----------->onStart");
    }

    public void onRestart()
    {
     super.onStart();
     Log.d(TAG," B----------->onRestart");
    }

    public void onResume()
    {
     super.onResume();
     Log.d(TAG," B----------->onResume");
    }
```

```
    public void onPause()
    {
       super.onPause();
       Log.d(TAG," B----------->onPause");
    }

    public void onStop()
    {
       super.onStop();
       Log.d(TAG," B----------->onStop");

    }

    public void onDestroy()
    {
       super.onDestroy();
       Log.d(TAG," B----------->onDestroy");

    }
}
```

同样的，ActivityB 的 Button 单击事件是从 ActivityB 跳转到 ActivityA，这样 ActivityB 也会经历一部分的生命周期。

通过以上两个 Activity 的代码，我们可以观察日志，以进一步理解 Activity 的生命周期。运行以上代码，当程序启动时，观察到日志如图 8.12 所示。

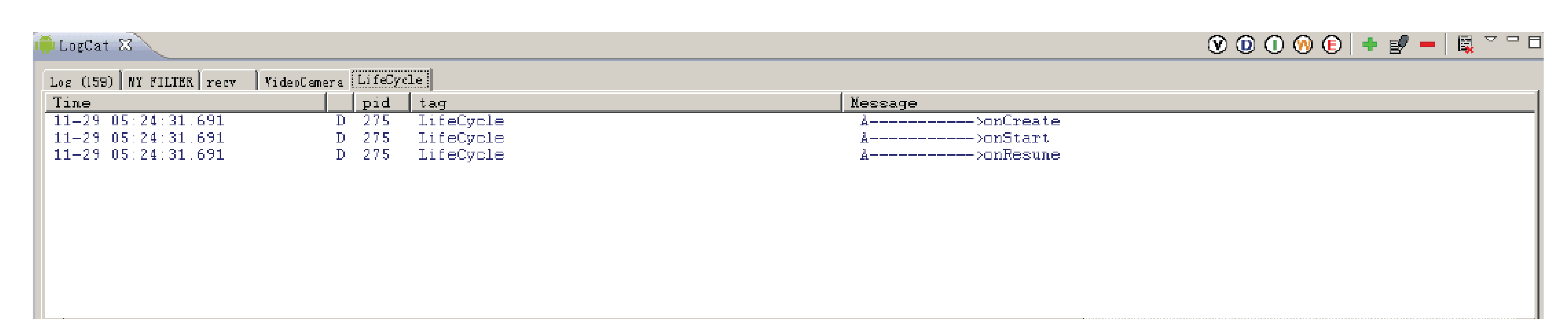


图 8.12　ActivityA 启动

从日志得出结论，ActivityA 在程序启动时一共经历了 3 个生命阶段，分别是 onCreate()、onStart()以及 onResume()，这与我们之前的分析不谋而合。此时 ActivityA 已经可以进行用户操作了，所以我们可以单击 Button 按钮以实现 Activity 的转换单击按钮后，日志输出如图 8.13 所示。

Time		pid	tag	Message
11-29 05:24:31.691	D	275	LifeCycle	A----------->onCreate
11-29 05:24:31.691	D	275	LifeCycle	A----------->onStart
11-29 05:24:31.691	D	275	LifeCycle	A----------->onResume
11-29 09:35:15.171	D	275	LifeCycle	A----------->onPause
11-29 09:35:15.281	D	275	LifeCycle	B----------->onCreat
11-29 09:35:15.321	D	275	LifeCycle	B----------->onStart
11-29 09:35:15.321	D	275	LifeCycle	B----------->onResume
11-29 09:35:15.712	D	275	LifeCycle	A----------->onStop

图 8.13　ActivityA 跳转到 ActivityB

从日志上我们可以清晰地看到，当跳转动作执行时，首先 ActivityA 进行了暂停也就是 onPause()，然后 ActivityB 开始执行 onCreate()、onStart()以及 onResume()过程，最后 ActivityA 执行 onStop()方法。

同样地，当我们从 ActivityB 再次跳转到 ActivityA 时，日志如图 8.14 所示。

Time		pid	tag	Message
11-29 05:24:31.691	D	275	LifeCycle	A------------>onStart
11-29 05:24:31.691	D	275	LifeCycle	A------------>onResume
11-29 09:35:15.171	D	275	LifeCycle	A------------>onPause
11-29 09:35:15.281	D	275	LifeCycle	B------------>onCreat
11-29 09:35:15.321	D	275	LifeCycle	B------------>onStart
11-29 09:35:15.321	D	275	LifeCycle	B------------>onResume
11-29 09:35:15.712	D	275	LifeCycle	A------------>onStop
11-29 09:40:20.172	D	275	LifeCycle	B------------>onPause
11-29 09:40:20.321	D	275	LifeCycle	A------------>onCreate
11-29 09:40:20.321	D	275	LifeCycle	A------------>onStart
11-29 09:40:20.321	D	275	LifeCycle	A------------>onResume
11-29 09:40:20.720	D	275	LifeCycle	B------------>onStop

图 8.14　从 ActivityB 跳转到 ActivityA

我们看到其生命周期的执行过程与之前是完全一样的，这里就不再赘述。那么到这里，Activity 依然没有执行销毁过程，也就是系统还没有执行 onDestroy()方法。不要着急，按下手机的 back 键，程序会回到上一个 Activity，也就是 ActivityB，并销毁 ActivityA，那么看看输出的日志吧，如图 8.15 所示。

Time		pid	tag	Message
11-29 09:35:15.712	D	275	LifeCycle	A------------>onStop
11-29 09:40:20.172	D	275	LifeCycle	B------------>onPause
11-29 09:40:20.321	D	275	LifeCycle	A------------>onCreate
11-29 09:40:20.321	D	275	LifeCycle	A------------>onStart
11-29 09:40:20.321	D	275	LifeCycle	A------------>onResume
11-29 09:40:20.720	D	275	LifeCycle	B------------>onStop
11-29 09:42:59.334	D	275	LifeCycle	A------------>onPause
11-29 09:42:59.440	D	275	LifeCycle	B------------>onRestart
11-29 09:42:59.440	D	275	LifeCycle	B------------>onStart
11-29 09:42:59.450	D	275	LifeCycle	B------------>onResume
11-29 09:42:59.821	D	275	LifeCycle	A------------>onStop
11-29 09:42:59.821	D	275	LifeCycle	A------------>onDestroy

图 8.15　按下 back 键从 ActivityA 回到 ActivityB

分析一下日志，我们会发现，按下 back 键后，首先 ActivityA 被暂停，接着 ActivityB 执行了 onRestart()接着是 onStart()和 onResume()，最后 ActivityA 被停止并销毁。为什么这里 ActivityB 没有执行到 onCreate()呢？因为之前 ActivityB 的资源没有经历 onDestroy()，所以并不需要再次 onCreate()，只需执行 onRestart()再次唤醒就可以了。

再次按下 back 键，我们会发现系统再次执行了相同的生命周期过程，这里就不再赘述。接下来，让我们想一想，当按下系统的 home 键后，Activity 的生命周期如何呢？我们依然观察日志，如图 8.16 所示。

Time		pid	tag	Message
11-29 09:42:59.440	D	275	LifeCycle	B------------>onStart
11-29 09:42:59.450	D	275	LifeCycle	B------------>onResume
11-29 09:42:59.821	D	275	LifeCycle	A------------>onStop
11-29 09:42:59.821	D	275	LifeCycle	A------------>onDestroy
11-29 09:51:29.581	D	275	LifeCycle	B------------>onPause
11-29 09:51:29.650	D	275	LifeCycle	A------------>onRestart
11-29 09:51:29.650	D	275	LifeCycle	A------------>onStart
11-29 09:51:29.650	D	275	LifeCycle	A------------>onResume
11-29 09:51:30.021	D	275	LifeCycle	B------------>onStop
11-29 09:51:30.021	D	275	LifeCycle	B------------>onDestroy
11-29 09:51:37.880	D	275	LifeCycle	A------------>onPause
11-29 09:51:38.670	D	275	LifeCycle	A------------>onStop

图 8.16　按下 home 键回到桌面

我们看到 ActivityA 只执行了两个过程：onPause()以及 onStop()，当我们长按 home 键再次进入程序，按照我们的推理，Activity 应该执行的生命周期流程应该是：

```
onRestart()—>onStart()—>onResume()
```

试一下，看看是不是推理正确呢？日志如图 8.17 所示。

Time		pid	tag	Message
11-29 09:42:59.821	D	275	LifeCycle	A------------>onDestroy
11-29 09:51:29.581	D	275	LifeCycle	B------------>onPause
11-29 09:51:29.650	D	275	LifeCycle	A------------>onRestart
11-29 09:51:29.650	D	275	LifeCycle	A------------>onStart
11-29 09:51:29.650	D	275	LifeCycle	A------------>onResume
11-29 09:51:30.021	D	275	LifeCycle	B------------>onStop
11-29 09:51:30.021	D	275	LifeCycle	B------------>onDestroy
11-29 09:51:37.880	D	275	LifeCycle	A------------>onPause
11-29 09:51:38.670	D	275	LifeCycle	A------------>onStop
11-29 09:57:40.401	D	275	LifeCycle	A------------>onRestart
11-29 09:57:40.401	D	275	LifeCycle	A------------>onStart
11-29 09:57:40.420	D	275	LifeCycle	A------------>onResume

图 8.17　执行生命周期的过程

没有错！根据日志，系统果然按照我们期望地那样执行了生命周期过程。那么到这里，相信读者朋友们应该能理性地理解 Activity 的生命周期了。如果还有疑问，可以多运行一下我们的范例程序，通过观察日志仔细揣摩一下，所谓书读百遍其义自现，多思考一下，相信没有问题的！

8.2　使用广播接收器

在 8.1 节中我们学习了使用 Intent 进行 Activity 间的跳转，实际上该 Intent 又被称为直接 Intent，因为它指定了要跳转的目标，此时 Android 系统不需解析，只要直接启动目标 Activity 就可以了。既然有直接 Intent 就肯定有间接 Intent，间接 Intent 是指一个没有指定具体目标的 Intent，只是在本身被创建时添加了一些描述信息，如种类、动作等。这类 Intent 通常会被“广播”出去，所有关心该 Intent 的广播接收器都会接受广播并处理。

8.2.1　发送广播

前文我们曾经说明，一个间接 Intent 必须要附带相关的“说明信息”，只有根据这些说明信息，系统才能正确解析一个 Intent 并将之发送到正确的接收者。一个 Intent 包含的说明信息如下：

（1）Action：操作，要执行的动作的定义。

（2）data：数据，指定动作相关联的数据。

（3）type：数据类型，指定动作的数据类型。

（4）categoy：类别，对执行动作的附加信息。

（5）extras：附件信息，其他所有的附加信息。

（6）component：目标组件，指定目标组件。

接下来让我们详细地了解这些描述信息。

1. Action

Android 中有许多预定义的标准 Action。例如，我们最熟悉的 ACTION_MAIN，它的值是“android.intent.action.MAIN”，这个值我们经常在 AndroidManifest.xml 文件中看到。表示当前的 Activity 是程序的入口，所有系统定义的标准动作如表 8-2 所示。

表 8-2　系统标准Action

Action	含　义
ACTION_MAIN	程序主入口
ACTION_VIEW	向用户显示数据，通常和特定的数据配合使用
ACTION_ATTACH_DATA	关联数据动作，比如将头像关联到联系人
ACTION_EDIT	编辑特定数据的操作
ACTION_PICK	从一组数据中进行选择操作
ACTION_CHOOSER	显示一个 Activity 选择器以供用户选择
ACTION_GET_CONTENT	让用户选择一类数据

续表

Action	含　　义
ACTION_DIAL	给用户打电话，配合指定的 data 可以触发拨出电话
ACTION_CALL	指定打电话给默认的动作，ACTION_CALL 在应用中启动一次呼叫有缺陷，多数应用 ACTION_DIAL
ACTION_SEND	给某人发送信息
ACTION_SENDTO	根据数据发送信息给指定的人
ACTION_ANSWER	处理来电
ACTION_INSERT	执行插入数据操作
ACTION_DELETE	执行删除数据操作
ACTION_RUN	运行数据，不管它意味着什么
ACTION_SYNC	指定一次数据同步
ACTION_ PICK_ACTIVITY	与 ACTION_CHOOSER 类似，不过不能直接启动选中的 Activity
ACTION_SEARCH	执行一次搜索
ACTION_WEB_SEARCH	执行一次网络搜索
ACTION_FACTORY_TEST	手机在工厂测试模式下启动的程序主入口

2. data

要操作的数据，它是以 Uri 的形式表示的。关于 Uri 的详细内容会在下一章 Android 中的数据存储中进行讲解，这里只做大概的了解。以电话为例，以下是一组 action/data 数据表示的意义。

- ACTION_VIEW　content://contacts/1：显示电话簿中 id 为 1 的联系人的信息。
- ACTION_DIAL　content://contacts/1：呼叫电话簿中 id 为 1 的联系人。
- ACTION_VIEW　tel:123 ：显示号码为 123 的电话拨出界面。
- ACTION_DIAL　tel:123 ：根据号码 123 拨出电话。
- ACTION_EDIT　content://contacts/1：编辑电话簿中 id 为 1 的联系人的信息。
- ACTION_VIEW　content://contacts/：显示电话簿中所有联系人的信息。

3. category

执行动作的附加信息，表 8-3 列举了系统标准的类别。

表 8-3　系统标准执行动作附加信息的类别

Category	含　　义
CATEGORY_DEFAULT	执行默认操作时设置，在初始化 Intent 时基本不被使用，只有在 filter 中才设置
CATEGORY_BROWSABLE	如果一个 Activity 在浏览器中被安全地调用就必须有这个类别信息
CATEGORY_TAB	该 Intent 作为打开一个 TabActivity 的子 Tab 时被使用
CATEGORY_ALTERNATIVE	该类别表示当前的 Intent 是用户正在浏览的可选动作中的一个
CATEGORY_SELECTED_ALTERNATIVE	表示该 Intent 是用户正在浏览的可选动作的替代选择
CATEGORY_LAUNCHER	该 Activity 启动时显示在顶层
CATEGORY_INFO	显示包的信息
CATEGORY_HOME	桌面，当手机启动时的第一个 Activity
CATEGORY_PREFERENCE	表示该 Activity 是一个偏好面板

4. type

通过该属性显示地指明数据的类型。一般情况下 type 可以由数据本身进行判定，如果显示指定则免去了推导过程。

5. component

目标组件。一般情况下，目标组件由 Intent 的相关描述信息进行推导，如果设置了目标组件则不再进行推导过程，直接打开目标组件。

6. Extras

Extras 在之前的学习中已经使用，可以使用它传递一些参数，如发送邮件时将邮件名、正文都添加到 Extras 中，再通过 Intent 传递给 E-mail 发送 Activity。

通过上述信息，我们就可以详细地构造一个间接 Intent 了。掌握新建一个间接 Intent 后，我们再来将这个 Intent“广播”出去！

发送广播非常简单，只需调用如下方法就可以了：

```
ContextWrapper.sendBroadcast(Intent intent)
```

这样一个新鲜出炉的 Intent 就被发送出去了。

8.2.2　接收广播

与现实生活中的广播一样，电台发送了广播，如果我们要收听就必须有一台收音机。Android 中的收音机叫做广播接收器——BroadcastReceiver。每一个广播接收器都必须有一个 Intent 过滤器，用来指定接收怎样的 Intent 广播。

要使用 BroadcastReceiver 需要如下 4 个步骤：

（1）新建一个 Intent 过滤器。

（2）新建一个 BroadcastReceiver。

（3）注册广播接收器。

（4）注销广播接收器。

让我们从第一步开始做起。一个 IntentFilter 指定了该广播接收器接收怎样的 Intent，那么它是怎样实现的呢？接下来我们就开始新建一个新的 IntentFilter：

（1）新建 IntentFilter

新建 IntentFilter 时使用构造方法：

```
IntentFilter.IntentFilter(String action)
```

这里的参数 action 就是 8.2.1 小节中讲解的 action 了，如果你还不太理解可以重新阅读上一小节。

接下来根据需要设置其他需要的属性。例如，要添加类别属性，可以选用方法：

```
IntentFilter.addCategory(String category)
```

更多属性设置与此大同小异，读者朋友们可以参考 API 文档，这里就不再赘述。

（2）新建广播接收器

在新建广播接收器时需要重写 onReceive()方法，具体代码如下：

```
BroadcastReceiver  receiver = new BroadcastReceiver()
        {
            @Override
            public void onReceive(Context ctx, Intent intent)
            {
                //接收到广播后执行的操作
            }
        };
```

这里需要注意的是，在 onReceive()方法中只能执行一些短时间的代码，一旦代码执行时间超过 5 秒就会出现超时对话框，所以最好一些耗时的操作放在一个线程里，或者放在一个 Activity 或者 Service 中，再通过 Intent 去启动他们。

（3）注册一个接收器

当广播接收器新建完成后并不能马上进入工作状态，因为 Android 系统还不知道你已经拥有了一个接收器。所以接下来我们还需要注册它，方法如下：

```
ContextWrapper.registerReceiver(BroadcastReceiver receiver, IntentFilter
filter)
```

这时我们就需要用到新建的 IntentFilter 了。通过该方法，你的广播接收器就可以正常接收广播了！

（4）当我们不再关注广播时，我们需要将接收器注销。方法如下：

```
ContextWrapper.unregisterReceiver(BroadcastReceiver receiver)
```

8.2.3　广播实例

接下来我们通过一个实例来完成对广播的实践，在实例中你可以学到如何发送一个广播，并在 Activity 中实现广播的接收，并在接收到广播后读取 Intent 携带的数据并显示在屏幕上。

1. 布局文件

在配置文件中添加如下 3 个组件，如表 8-4 所示。

表 8-4　组件类型

组件类型	ID	含　　义
TextView	Tv	显示接收到的 Intent 中的信息
Button	Btn1	发送一种广播
Button	Btn2	发送另一种广播

2. 整体设计

整体设计中，在 onCreate()方法中完成接收器的新建和注册，在初始化界面函数中完成按钮的单击事件，不同的按钮发送不同 Intent 的广播。

```
public class BroadcastDemo extends Activity {
```

```
    BroadcastReceiver receiver;
    public static final String ACTION 1 = "com.wes.action.BROADCAST 1";
    public static final String ACTION 2 = "com.wes.action.BROADCAST 2";
    Button btn1;
    Button btn2;
    TextView tv;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        initView();                                     //初始化界面

        receiver = new BroadcastReceiver()              //新建广播接收器
       {
              @Override
              public void onReceive(Context ctx, Intent intent)//接收回调函数
              {
                  String action = intent.getAction();//得到 Action
                  String data = intent.getExtras().getString("data");
                                                        //得到附带的信息
                  if (action.equals(ACTION_1))          //判断 Action 的种类
                  {
                       tv.setText("接收到：\n"+ACTION_1+"\n 内容是：\n"+data);
                  }
                  else if (action.equals(ACTION 2))
                  {
                       tv.setText("接收到：\n"+ACTION 2+"\n 内容是：
                       \n"+data);
                  }
                  else
                  {
                       //如果 Action 不是以上两个则表示我们不关心，不做操作
                  }
              }
       };
       IntentFilter filter1 = new IntentFilter(ACTION 1);
                                 //新建一个过滤器，接收 Action_1 的 Intent
       filter1.addAction(ACTION_2);                     //添加接收的 Action
       registerReceiver(receiver, filter1);             //注册广播接收器
    }
}
```

3．初始化界面函数

按下 btn1 时发送 Action 为 ACTION_1 的 Intent，按下 btn2 时发送 Action 为 ACTION_2 的 Intent。

```
public void initView()
   {
   btn1 = (Button)findViewById(R.id.btn1);
   btn2 = (Button)findViewById(R.id.btn2);
   tv   = (TextView) findViewById(R.id.tv);
```

```
    OnClickListener listener = new OnClickListener()
         {
              @Override
              public void onClick(View v)
              {
                   int id = v.getId();
                  switch(id)
                  {
                  case R.id.btn1:
                     Intent intent1 = new Intent(ACTION_1);
                                     //新建 Intent，Action 为 ACTION_1
                     intent1.putExtra("data", "我是 action_1");  //添加数据
                     sendBroadcast(intent1);                     //发送广播
                     break;
                case R.id.btn2:
                     Intent intent2 = new Intent(ACTION_2); //新建 Intent
                     intent2.putExtra("data", "我是 action_2");  //添加数据
                     sendBroadcast(intent2);                     //发送广播
                     break;
                default:
                     break;
                }
            }
        };

        btn1.setOnClickListener(listener);
        btn2.setOnClickListener(listener);

    }
```

4. Activity结束时，注销广播接收器

```
    public void onStop()
    {
          super.onStop();
          unregisterReceiver(receiver);                         //注销接收器
    }
```

运行以上代码，效果如图 8.18 所示。



程序初始化界面



按下 btn1 后的效果



按下 btn2 后的效果

图 8.18

8.3 使用服务

本节将介绍 Android 应用程序的另一重要组成部分——服务（Service）。服务可以看做是一个没有界面的 Activity。例如，音乐播放器，当我们的手机离开播放界面时，我们希望程序仍然运行在后台，这样就可以在听音乐的同时继续做其他的事情。

服务运行在后台，不可交互，它不可以自己启动，需要其他的 Context 执行 startService()开始服务或者执行 bindService()绑定服务。这两种启动服务的方法是有区别的，这在后面的讲解中会详细讨论。

8.3.1 新建服务

正如我们之前所说的一样，服务就类似于一个没有界面的 Activity，不同的是它们的存在意义和自身的生命周期。新建服务时，步骤如下：

（1）新建一个类，继承自 Service 类。

（2）重写 Service 的几个重要方法。

接下来就开始进入具体的实现步骤，首先在 Eclipse 的工程浏览器中选择新建一个工程，这一步大家必定了然于心，新建好一个工程后，再次单击右键，选择新建一个 Class，如图 8.19 所示。

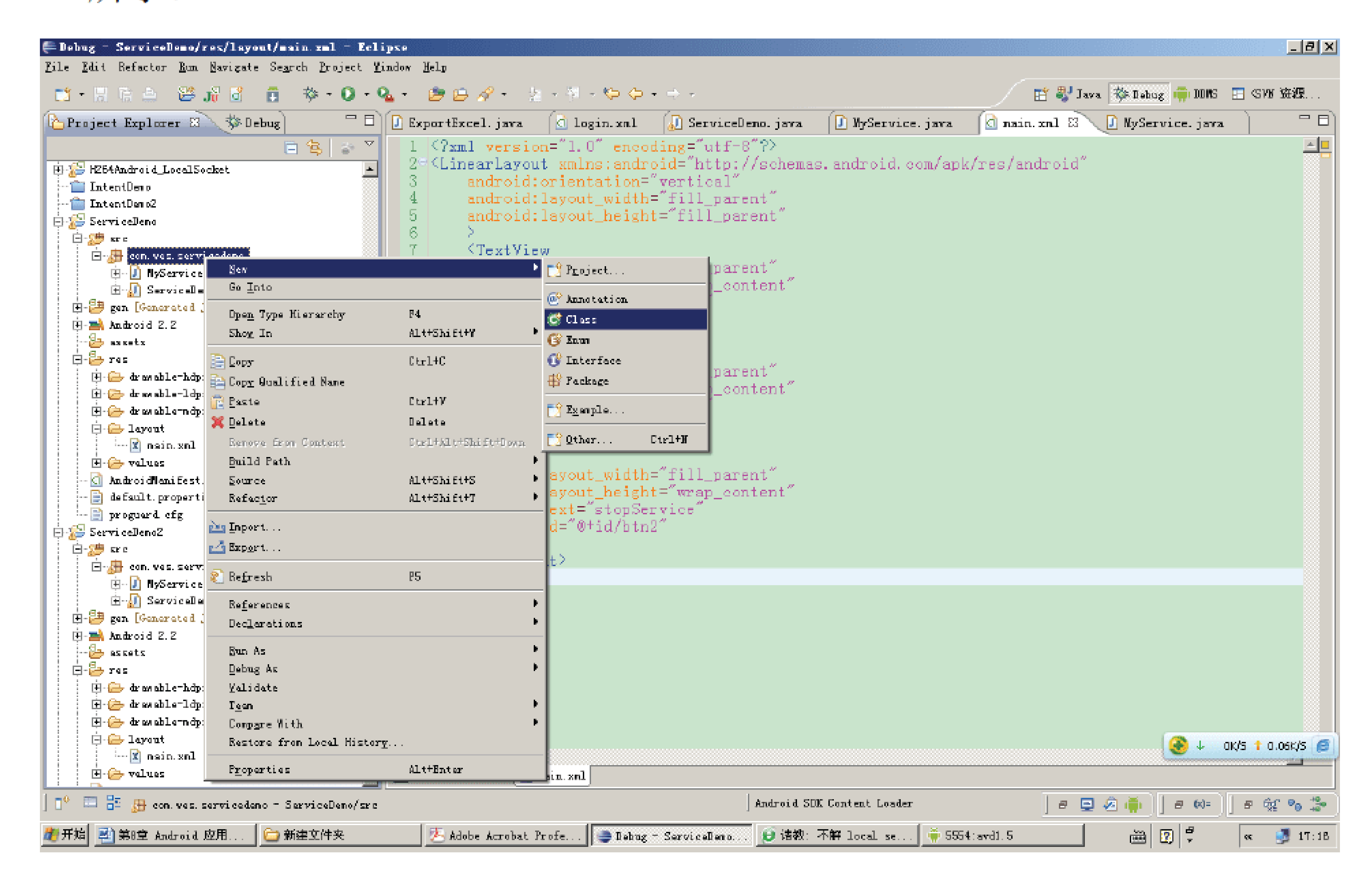


图 8.19　新建类

接着在弹出的新建 Java 类对话框中输入类名，选择父类（Superclass）为 android.app.Service，如图 8.20 所示。

New Java Class

Java Class

Create a new Java class.

Source folder: ServiceDemo/src　Browse...

Package: com.wes.servicedemo　Browse...

Enclosing type:　Browse...

Name: MService

Modifiers: public　default　private　protected

abstract　final　static

Superclass: android.app.Service　Browse...

Interfaces:　Add...　Remove

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

Generate comments

Finish　Cancel

图 8.20　选择继承自 Service

单击 Finish 按钮后，一个新的类就创建完毕了，接着在空白处单击右键，在菜单中选择 Source，在弹出的子菜单中选择重写/实现方法（Override/Implements Method），如图 8.21 所示。

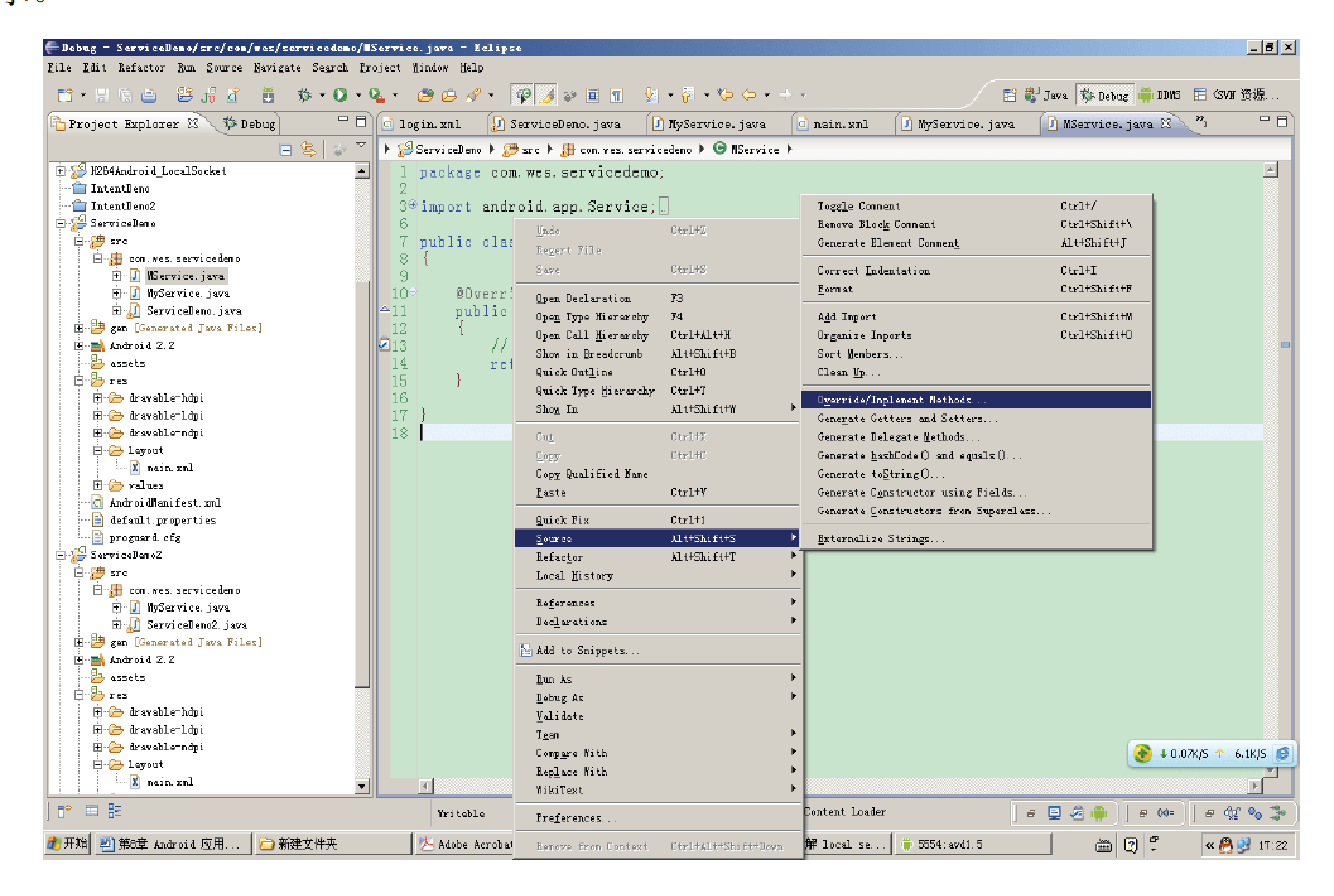


图 8.21　选择重写方法功能

选择该功能后，就进入了如图 8.22 所示重写方法对话框，在 Service 下选择：

（1）onCreate()

（2）onDestroy()

（3）onRebind()

（4）onStart()

（5）onUnbind()

连同在创建时就默认重写的 onBind()方法，我们一共需要完成 Service 类的 6 个重要方法，也就是它的 6 个生命周期。

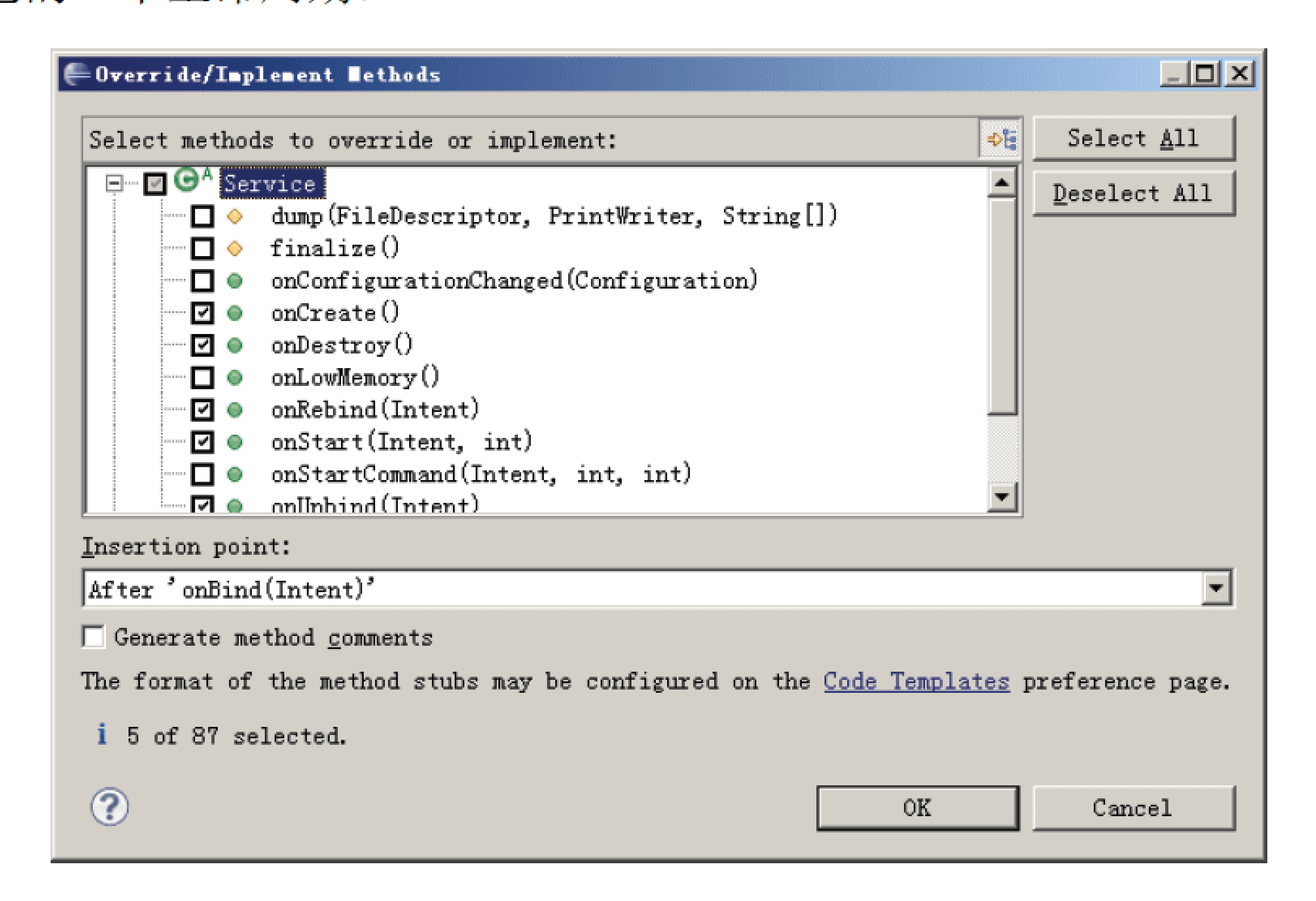


图 8.22　重写方法选择

此时我们新建的 MyService 类的代码应该如下所示：

```
package com.wes.servicedemo;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service
{
    @Override
    public IBinder onBind(Intent intent)
    {
        return null;
    }

    @Override
    public void onRebind(Intent intent)
    {
        super.onRebind(intent);
    }

    @Override
    public void onCreate()
    {
        super.onCreate();
```

```
    }

    @Override
    public void onStart(Intent intent, int startId)
    {
         super.onStart(intent, startId);
    }

    @Override
    public void onDestroy()
    {
         super.onDestroy();
    }

    @Override
    public boolean onUnbind(Intent intent)
    {
          return super.onUnbind(intent);
    }

}
```

接着我们注意观察 onBind()方法，我们发现它需要返回一个 Ibinder 类型的实例。这个对象的作用是负责 Service 和 Activity 通信的。实现代码如下：

```
private final IBinder binder = new MyBinder();

    public class MyBinder extends Binder
    {
        MyService getService()
        {
            return MyService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent)
    {
        Log.i(TAG,"=======>onBind");
        Toast.makeText(getBaseContext(), "onBind", Toast.LENGTH_SHORT).
        show();
        return binder;
    }
```

我们新建了一个继承自 Binder 的内部类，在内部类中写了一个方法，是 getService()，该方法的作用就是在 Activity 中得到 Service 的对象，这样就可以操作 Service 了。与 onBind()方法类似，为了能直观地看到 Service 方法成功调用与否，我们在每个方法中添加一行 Toast.makeText()方法。

8.3.2　使用 Service

在上一小节的内容中我们新建了一个 Service 类，并完成了其中需要重写的方法，接下来我们就开始启动 Service 吧！

在启动 Service 之前千万不要忘记了在 AndroidManifest 注册文件中为 Service 注册，否则系统将无法找到 Service。添加了 Service 的注册文件如下所示，注意 Service 添加的位置，

它是与 Activity 平级的。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.wes.servicedemo"
     android:versionCode="1"
     android:versionName="1.0">
   <uses-sdk android:minSdkVersion="8" />

   <application android:icon="@drawable/icon" android:label="@string/
   app name">
      <activity android:name="MainActivity"
               android:label="@string/app name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
           <!-- service 与 activity 平级 ，注意其添加的位置-->
       <service android:enabled="true"  android:name=".MyService"/>
   </application>
</manifest>
```

启动 Service 有两种方法，分别是：

```
ContextWrapper.startService(Intent service)
ContextWrapper.bindService(Intent service, ServiceConnection conn, int
flags)
```

先分析第一种方法：startService(intent)。使用这类方式启动 Service 可以使 Service 独立运行在后台，不受 Activity 生命周期的影响。也就是说，startService()执行后，即使所有的 Activity 都退出了，Service 仍然在运行，直到有 Context 对象调用 stopService()为止。

接下来就在代码中实现用 startService()启动我们之前新建的 Service。首先在布局文件中添加两个 Button，分别用来启动服务和停止服务。实现代码如下：

```
public class ServiceDemo extends Activity
{
      Button btn1;
      Button btn2;
      MyService mService;
      @Override
   public void onCreate(Bundle savedInstanceState)
    {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);

       btn1 = (Button) findViewById(R.id.btn1);
       btn2 = (Button) findViewById(R.id.btn2);

       btn1.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Intent i = new Intent(ServiceDemo.this,MyService.class);
                                                  //新建 Intent 对象
                startService(i);                  //开始 Service
            }
        });
```

```
            btn2.setOnClickListener(new OnClickListener()
             {
                 @Override
                 public void onClick(View v)
                 {
                     Intent i = new Intent(ServiceDemo.this,MyService.class);
                                                            //新建 Intent 对象
                     stopService(i);                        //停止 Service
                 }
             });
          }
      }
```

我们发现代码实现非常方便，与 startActivity()方法相同。那么运行一下效果如何呢？单击 startService 按钮，显示效果如图 8.23 所示。



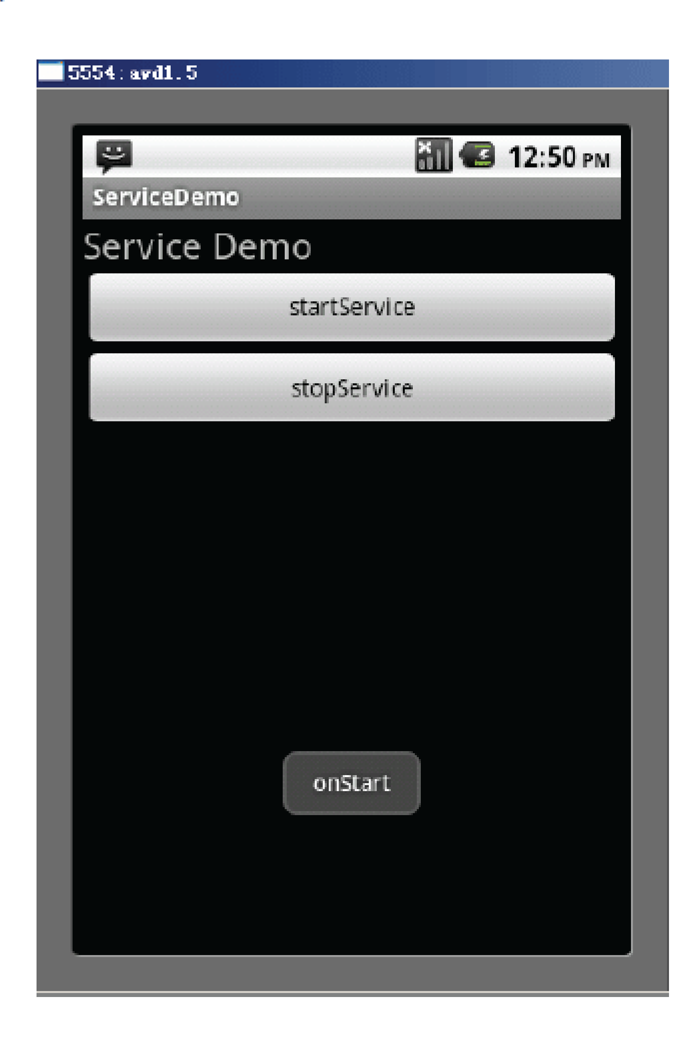


图 8.23　开始服务

接着，单击 stopService 按钮，就可以停止服务了，此时显示如图 8.24 所示。



图 8.24　结束服务

成功使用 startService()方法启动服务后，我们接着使用另一种方法启动 Service，就是 bindService()方法了。

使用 bindService()方法就是将 Service 与 Activity 绑定起来，一旦 Activity 结束则 Service 同时结束。这种生存状态有些类似于“桃园三结义”中的“不求同年同月同日生，但求同年同月同日死”。使用 bindService()方法启动 Service 相较于 startService()方法略麻烦一些，因为我们还需要一个 ServiceConnection 对象。这个对象的作用是实现 Activity 与 Service 的绑定，实现方法如下：

```
    MyService mService;
ServiceConnection  mConnection = new ServiceConnection()
    {
        @Override
        public void onServiceDisconnected(ComponentName name)
        {
            mService = null;                             //连接断开时
        }
        @Override
        public void onServiceConnected(ComponentName name, IBinder service)
        {
            mService = ((MyService.MyBinder) service).getService();
    //连接成功时
        }
    };
```

在新建 ServiceConnection 对象时需要完成 ServiceConnection 接口的两个方法，分别是 onServiceDisconnected 以及 onServiceConnected。顾名思义，这两个方法分别表示断开连接和连接成功。在断开连接时我们将 MyService 对象设为空，在连接成功时为 MyService 得到其操作对象。事实上，在连接 Service 时，会调用 onBind()方法，在新建 Service 时我们就重写了这个方法，在该方法中我们会返回一个 Ibinder 对象，这个对象就是 onServiceConnected 方法中的第二个参数。接着我们将 Ibinder 类型转换为我们之前写的 MyBiner 内部类对象，再通过这个对象的 getService()方法就可以得到 Service 的操作对象。

接着我们就可以开始进行服务的绑定了，其代码段如下所示：

```
    btn1 = (Button) findViewById(R.id.btn1);
           btn2 = (Button) findViewById(R.id.btn2);

           btn1.setOnClickListener(new OnClickListener()
            {
                @Override
                public void onClick(View v)
                {
                    Intent i = new Intent(ServiceDemo2.this,MyService.class);
                    bindService(i,mConnection,Context.BIND AUTO CREATE);
                                                              //绑定服务
                }
            });

           btn2.setOnClickListener(new OnClickListener()
            {
                @Override
                public void onClick(View v)
                {
                    unbindService(mConnection);               //解除绑定
                }
            });
```

接着我们就可以运行程序了，运行后效果如图 8.25 所示。





图 8.25　绑定服务

单击 unBindService 按钮，解除绑定。效果如图 8.26 所示，表明成功解除绑定。





图 8.26　解除绑定

本小节我们学习了 Service 的启动方法，那么我们应该怎样管理它的生命周期呢？不同的启动方法它的执行流程又是怎样的呢？这些问题在下一小节我们将给出详细的解答。

8.3.3　Service 的生命周期

现在我们已经学习了两种启动 Service 的方法，我们发现每次启动时都需要执行

onCreate()，在结束时都需要执行 onDestroy()方法。这就是 Service 的生命周期的一个重要的特点，本小节我们将通过日志文件来揭开 Service 生命周期的神秘面纱。

1. startService()方法生命周期

如果通过 startService()方法启动服务，则它的生命周期如下：

onCreate()——> onStart()——>onDestroy()

这就是一个服务完整的生命流程，如果重复 startService()，则不会进入 onCreate()方法，而是直接调用 onStart()方法。

2. bindService()方法生命周期

如果通过 bindService()方法启动服务，则其生命周期如下所示：

onCreate()——> onBind()——> onUnbind()——> onDestroy()

同样地，开始必须执行 onCreate()方法，接着执行 onBind()方法进行绑定。结束时，首先结束绑定 onUnbind()，再销毁服务 onDestroy()。

那么如果是 startService()和 bindService()两种方法交叉调用呢？我们接下来就通过一个实例来深入探究！

我们新建一个工程，首先将之前新建的 MyService 类复制到本工程，接着在工程中新建一个 MainActivity 类。

在布局文件中添加 3 个 Button 按钮，如表 8-5 所示。

表 8-5　MainActivity组件表

类　　型	ID	意　　义
Button	Btn1	startService
Button	Btn2	stopService
Button	Btn3	跳转到 NewActivity

在 MainActivity 中添加如下代码段：

```
@Override
   public void onCreate(Bundle savedInstanceState)
    {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.layout);

       btn1 = (Button) findViewById(R.id.btn1);
       btn2 = (Button) findViewById(R.id.btn2);
       btn3 = (Button) findViewById(R.id.btn3);

       btn1.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Intent i = new Intent(MainActivity.this,MyService.class);
                startService(i);                //startService()
            }
        });

       btn2.setOnClickListener(new OnClickListener()
```

```
            {
                @Override
                public void onClick(View v)
                {
                    Intent i = new Intent(MainActivity.this,MyService.class);
                    stopService(i);              //stopService()
                }
            });

        btn3.setOnClickListener(new OnClickListener()
            {
                @Override
                public void onClick(View v)
                {
                    Intent i = new Intent(MainActivity.this,NewActivity.class);
                    startActivity(i);           //前往 NewActivity
                }
            });
    }
```

接着我们新建另一个 Activity，名为 NewActivity，在该 Activity 中进行 bindService()方法和 unbindService()方法。

首先，在布局文件中添加 3 个 Button 按钮，如表 8-6 所示。

表 8-6　NewActivity组件表

类　　型	ID	意　　义
Button	Btn1	bindService
Button	Btn2	unbindService
Button	Btn3	跳转到 MainActivity

Java 部分代码段如下所示：

```
    public void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);

            mConnection = new ServiceConnection()
            {
              @Override
             public void onServiceDisconnected(ComponentName name)
             {
                  mService = null;
             }
             @Override
             public void onServiceConnected(ComponentName name, IBinder service)
             {
                  mService = ((MyService.MyBinder) service).getService();
             }
            };

        btn1 = (Button) findViewById(R.id.btn1);
        btn2 = (Button) findViewById(R.id.btn2);

        btn1.setOnClickListener(new OnClickListener()
            {
                @Override
                public void onClick(View v)
```

```
                {
                    Intent i = new Intent(NewActivity.this,MyService.class);
                    bindService(i,mConnection,Context.BIND AUTO CREATE);
                                                                //绑定服务
                }
            });

            btn2.setOnClickListener(new OnClickListener()
            {
                @Override
                public void onClick(View v)
                {
                    unbindService(mConnection);     //解除绑定
                }
            });
        }
```

最后，不要忘记了在注册文件中添加 NewActivity 以及 Service 的注册工作，这里就不再给出代码了，请参考 8.3.2 小节中的例子。

我们通过如下步骤进行 Service 生命周期的研究。

1．先开始服务后绑定服务

先开始服务——startService()，接着绑定服务——bindService()。我们观察其执行流程，如图 8.27 所示。

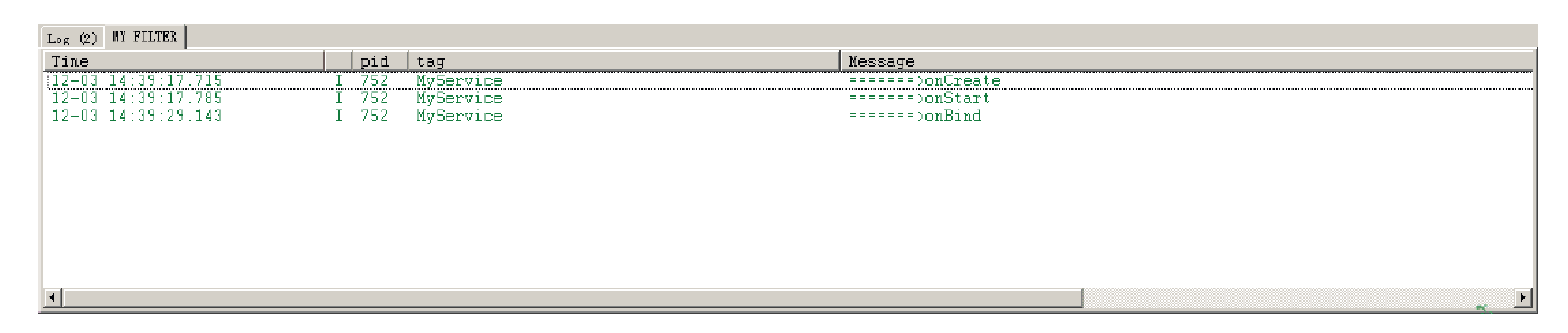


图 8.27　开始服务

根据日志，我们发现它的生命周期为：

onCreate()——> onStart()——> onBind()

也就是说，在 bindService()时，如果 Service 已经执行 onCreate()方法，则不会重复调用。事实上，Service 有一个原则：Service 只能执行一次 onCreatev()，不会重复创建。

2．先解除绑定后停止服务

此时我们单击“返回到 MainActivity”按钮，单击 stop 按钮，我们会发现无法正常结束服务，我们必须在 NewActivity 中先解除绑定，才能正常结束。也就是说：服务被绑定后不能直接停止。

值得一提的是，如果服务既是通过 startService()方法启动，之后又被绑定，那么在解除绑定时不会销毁服务。

单击 unbindService 按钮时日志如图 8.28 所示。

接着跳转到 NewActivity，单击 stopService 按钮，这样才能正常结束服务。此时，日志如图 8.29 所示。

Time		pid	tag	Message
12-03 14:52:28.653	I	814	MyService	=======>onCreate
12-03 14:52:28.694	I	814	MyService	=======>onStart
12-03 14:52:33.332	I	814	MyService	=======>onBind
12-03 14:52:36.083	I	814	MyService	=======>onUnBind

图 8.28　解除绑定

Time		pid	tag	Message
12-03 14:52:28.653	I	814	MyService	=======>onCreate
12-03 14:52:28.694	I	814	MyService	=======>onStart
12-03 14:52:33.332	I	814	MyService	=======>onBind
12-03 14:52:36.083	I	814	MyService	=======>onUnBind
12-03 14:55:52.504	I	814	MyService	=======>onDestroy

图 8.29　结束服务

3. 先绑定服务后启动服务

接下来我们再来看先绑定服务——bindService()，在开始服务——startService()，它们执行的流程是怎样的呢？如图 8.30 所示。

Time		pid	tag	Message
12-03 15:05:21.013	I	814	MyService	=======>onCreate
12-03 15:05:21.044	I	814	MyService	=======>onBind
12-03 15:05:28.515	I	814	MyService	=======>onStart
12-03 15:05:31.954	I	814	MyService	=======>onStart

图 8.30　先绑定服务

通过观察我们得到结论，其执行流程为：

onCreate()——> onBind()——> onStart()

如果重复 startService 则重复执行 onStart()方法。

结束服务时同样必须先解除绑定，之后才能正常结束服务。

结束时需要注意的是：在 MainActivity 界面时不能使用 Go To NewActivity 跳转到 NewActivity，因为如果是这样进入 NewActivity，此时的 NewActivity 与刚才执行 bindService()方法的 NewActivity 是不同的对象，此时的 NewActivity 根本没有执行 bindService()方法，如果直接单击 unbindService 按钮，会出现状态异常。所以我们要使用 back 键返回到之前的 NewActivity，此时执行 unbindService 才能正常解除绑定。

8.4　使用 ContentProvider

ContentProvider 是 Android 中处理数据相关的一个类。我们知道在 Android 中，所有

的数据都是私有的。也就是说每个不同程序拥有其独立的数据库、独立的数据存储方式。这保证了程序的私有性和安全性，但也给数据共享带来了一定的影响。那么难道在 Android 中就不能实现数据的共享吗？显然不可能，解决这个问题的办法就是使用 ContentProvider！

ContentProvider 类实现了一组标准的接口，一个程序可以通过实现 ContentProvider 的接口将自己的数据共享给其他程序，其他程序可以通过这些接口对数据进行查询、插入、删除以及修改等操作。

这里就先做一个简单的介绍，在第 9 章 Android 的数据存储中会有 ContentProvider 的详细讲解。

8.5　小　　结

本章讲解了 Android 应用程序的 4 个重要组成部分：Activity、BroadcastReceiver、Service 以及 ContentProvider。其中 ContentProvider 只做了简单介绍，下一节中会进行详细讲解。本章的重点是 Intent 以及广播的结合使用，难点是 Service 的生命周期的控制，尤其是两种方式交叉使用的时候。通过本章的学习，我们的应用程序的结构就可以更丰富，不再完全局限于 Activity 了。

下一章我们将讲解 Android 中数据存储的相关知识，将会是非常重要的一章。

第 9 章　Android 中的数据存储

数据存储可以帮助我们将需要的数据保存，以便需要的时候提取。数据存储的方式宏观上有两种：本地存储和网络存储。在这一章将着重讲解 Android 中的本地数据存储。Android 提供了 3 种操作数据的方式，即 SharedPreferences（共享首选项）、文件存储以及 SQLite 数据库。在本章中，除了需要学会这 3 种本地存储方式外，还需学习 ContentProvider（内容提供者）。

9.1　使用 SharedPreferences

SharedPreferences 可以帮助用户很快地保存一些数据项，并共享给当前应用程序或者其他应用程序。这给用户保存数据带来了很大的便利——再也不用为将这类数据保存在哪里更方便、更高效而发愁了。那么，在这一节中我们将学会使用它。在使用它之前，需要先了解它的存储格式、存储位置，要做到知其然，更知其所以然。

9.1.1　什么是 SharedPreferences

1．它可以保存哪些数据

SharedPreferences 是在 Android 中用来存储一些轻量级数据的，如一些开机欢迎语、用户名、密码等。它位于 Activity 级别，并可以被该程序的所有 Activity 共享。它支持的数据类型包括：布尔型（Boolean）、浮点型（Float）、整型（Int）、长整型（Long）和字符串（String）。一般情况下，程序员并不需要知道这些数据保存在哪里，又以什么方式保存。但这里为了更深入的了解 SharedPreferences，我们会做进一步的理解。

2．数据被保存在哪里了

SharedPreferences 保存的数据都存储在 Android 文件系统目录中的/data/data/PACKAGE_NAME/shared_prefs 下的 xml 文件中（这里仅以 Content.xml 为例，事实上该文件的名字是通过编程人员指定的，文件内容同理）。通过使用文件浏览器可以查看并导出文件，如图 9.1 和图 9.2 所示。

3．保存成什么样子

在 SharedPreferences 这种存储方式中，数据都是以“键-值”对的方式保存，这一点和 Map 很相似。如果读者对 Map 有一些了解，理解本节会非常轻松；如果读者对 Map 或者

HashMap 不是很了解，也没有关系，相信通过本小节的学习会拥有一个比较形象的认识。接下来我们可以看一下具体的保存形式。导出 Content.xml 文件后打开，如图 9.3 所示。

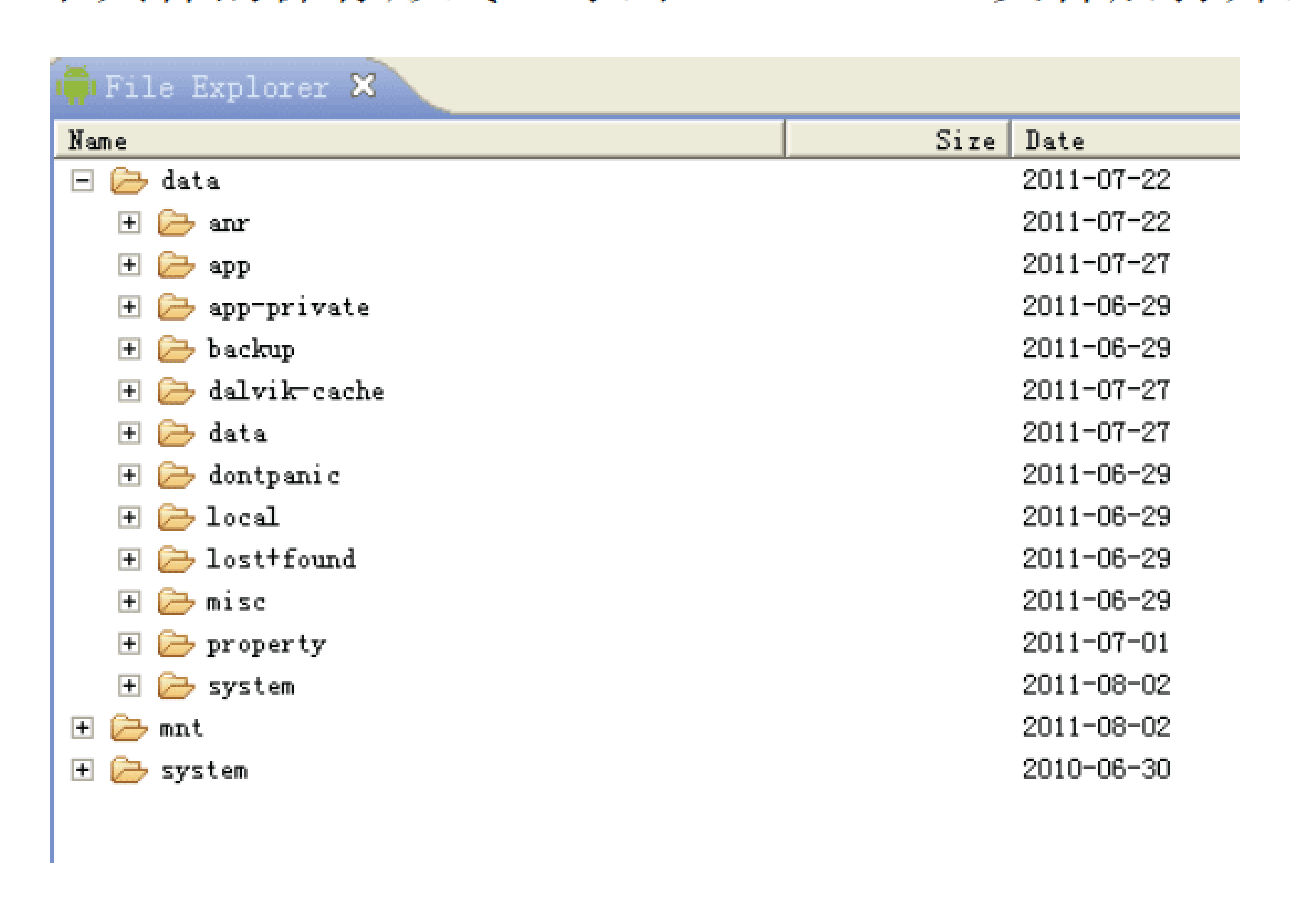


图 9.1　文件浏览器中的文件层级目录

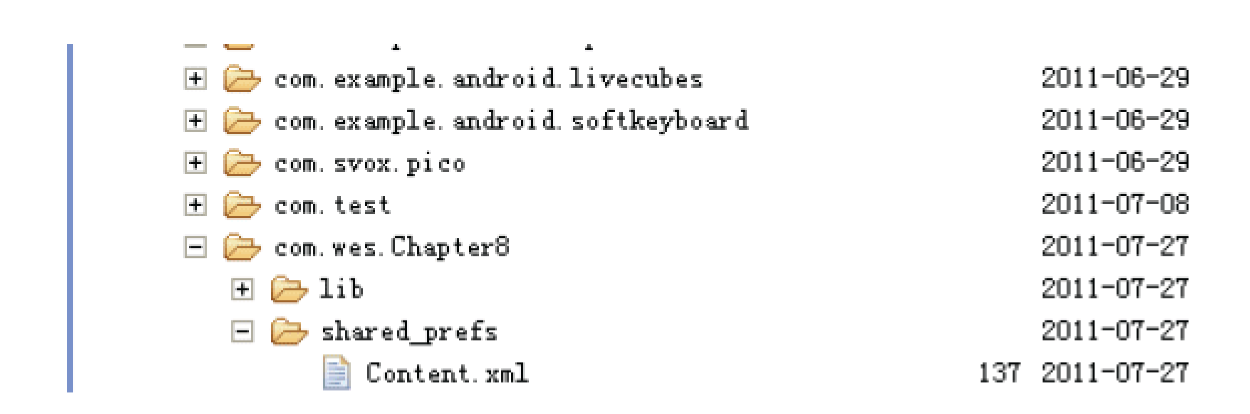


图 9.2　位于 data/data/com.wes.Chapter8/shared_prefs/下的 Content.xml 文件

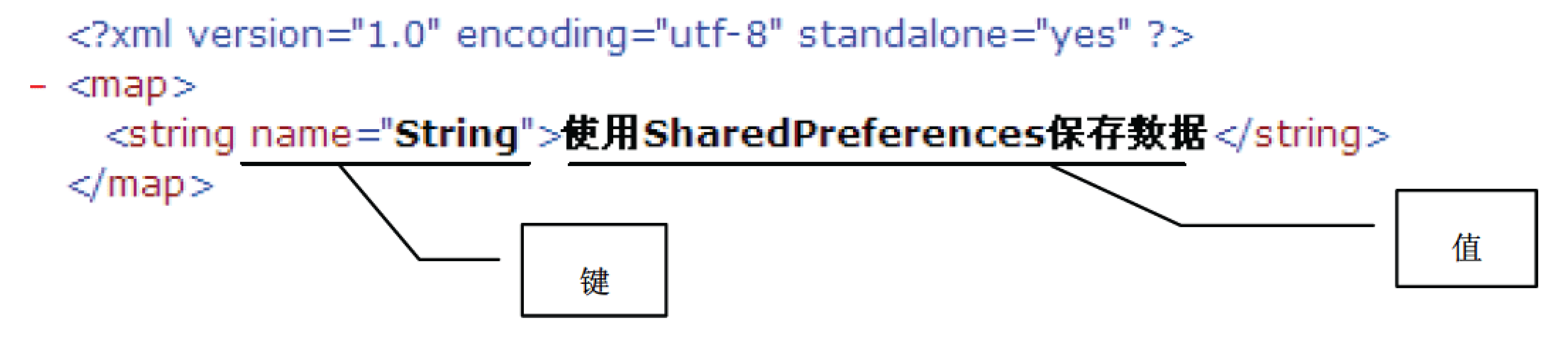


图 9.3　content.xml 文件中的内容

图 9.3 就是 SharedPreferences 的具体保存形式了，在 Map 节点下的内容都是我们存放进去的数据。

系统保存的 xml 文件在处理时，会由系统通过底层自带的本地 xml 解析器解析，如 XMLpull 等方式。这样的方式对于内存资源的占用比较低，数据量小的时候读取效率较高。而这种存储方式的不足是只能存储轻量级的数据，一旦数据比较大的时候，效率会成为一个很大的难题。

9.1.2　使用 SharedPreferences 保存数据

使用 SharedPreferences 保存数据要经过 4 个步骤：获取对象、创建编辑器、修改内容、提交修改。接下来就仔细查看这 4 个步骤究竟是怎样的一个过程。

1．获取对象

通过 getSharedPreferences()方法获取一个 SharedPreferences 对象，以方便对其进行相关操作，方法如下：

```
getSharedPreferences("Content", Context.MODE_PRIVATE);
```

在这里，第一个参数是保存的 SharedPreferences 的 TAG，即名称。第二个参数是这个 SharedPreferences 的应用模式，这里使用了 Context.MODE_PRIVATE 私有模式，这种模式代表该文件是私有数据，只能被应用程序本身访问。在该模式下，写入的内容会覆盖原文件的内容。事实上，还有 3 种模式可供使用，分别是：

- Context.MODE_APPEND：在这种模式下，系统会检查该文件是否存在。如果存在就往文件里追加内容，否则就创建一个新的文件以供保存。
- MODE_WORLD_READABLE：在这种模式下，当前文件可以被其他应用程序读取。
- MODE_WORLD_WRITEABLE：在这种模式下，当前文件可以被其他应用程序写入。

2．创建一个Editor编辑器

在 SharedPreferences 中要编辑信息，必须取得一个编辑器，也就是 Editor。Editor 对象的作用是提供一些方法以便使用者修改 xml 文件中的内容，如添加字符串或整数等。方法如下：

```
SharedPreferences.edit();
```

只要使用简单的 SharedPreferences.edit()方法就可以得到一个 editor 对象了。接下来你就可以使用这个对象去操作数据了。

3．使用Editor修改内容

在 9.1.1 小节中已经知道 SharedPreferences 的具体存储方式，那怎么向 xml 文件添加内容呢？这个时候就要使用 putString()方法了。这个方法是向 xml 文件中添加一个节点。SharedPreferences 根据方法名创建一个<String></String>节点，根据这个方法的参数向节点中添加内容。方法如下：

```
putString("String", data);
```

这里第一个参数是“键”，也就是所谓的 Key，第二个参数是“值”，也就是所谓的 Value。前文说过：在 SharedPreferences 中，所有的数据都是以“键-值”对的形式保存。这里的键是用来检索的索引，而值就是保存的对象。更形象的说法是：Key 相当于一个是章节名称，而 Value 是该章节下的内容。

修改的方法包括：

```
SharedPreferences.Editor.putString()
                              //向 SharedPreferences 中添加 string 类型数据
SharedPreferences.Editor.putBoolean()
                              //向 SharedPreferences 中添加 boolean 类型数据
```

```
SharedPreferences.Editor.putFloat()
                                    //向 SharedPreferences 中添加 float 类型数据
SharedPreferences.Editor.putInt()
                                    //向 SharedPreferences 中添加 int 类型数据
SharedPreferences.Editor.putLong()
                                    //向 SharedPreferences 中添加 long 类型数据
```

当然，还可以使用 SharedPreferences.Editor.clear()来清除所有的首选项，使用 SharedPreferences.Editor.remove()来移除指定的首选项。

4．提交内容

将数据修改好之后，也就是 putString()或其他 put()方法执行完后，要将这个修改提交给 SharedPreferences，以通知其将内容写入到 xml 文件中。使用的方法如下：

```
editor.commit();
```

注意，如果不提交，Android 是不会进行任何读写操作的。一些有过 Java 编程经验的程序员经常会犯这样的错误，因为在 Java 中，map.put()操作后，键和值就已经存放到了 Map 中。而 Android 一定要提交才能生效，这一点大家要牢记！

提交后在 xml 文件的<String></String>中会添加一些内容。例如：

```
<String name = "String"> 使用 SharedPreferences 保存数据</String>
```

其中 name =“String”是键或者说是 Key，而“使用 SharedPreferences 保存数据”这个字符串就是值，也就是 Value 了。而一个“键-值”对，也称之为一个映射。

9.1.3　使用 SharedPreferences 读取数据

通过以上四个步骤，就可以在你的程序中使用 SharedPreferences 了。那么把数据提交之后要使用的时候又要经过几个流程呢？不要着急，使用的时候就不那么麻烦啦，只要两个步骤就可以顺利取出保存的数据并使用。

1．获得SharedPreferences对象

同保存数据一样，要获得之前写入的数据，必须先获得一个你需要操作的 SharedPreferences 对象，获得了这个对象才能对相应的 SharedPreferences 进行操作。获得的方法如下：

```
getSharedPreferences("Content", Context.MODE_PRIVATE);
```

显而易见，这里第一个参数用来指定你的 SharedPreferences 名，也就是你要操作的 SharedPreferences；第二个参数是模式名，意义和前文介绍的一样。

2．取出Key对应的Value即内容

明白了 xml 存储方式后，我们知道 SharedPreferences 会从一个节点找到该节点中的内容并 return 给使用者，我们只要使用 getString()等方法就可以了：

```
SharedPreferences.getString()
SharedPreferences.getBoolean()
SharedPreferences.getFloat()
SharedPreferences.getInt()
SharedPreferences.getLong()
```

各个方法的意义与前文介绍的一一对应，笔者就不再赘言。

最后 SharedPreferences.getAll()获得所有的“键-值”对。

9.1.4　通过实例学习 SharedPreferences

接下来看一个小例子：该实例演示了基本的使用 SharedPreferences 保存和读取数据的操作。先看效果图，如图 9.4 和图 9.5 所示。我们一共使用了 3 类控件：





图 9.4　单击“注册”按钮之后，保存数据到 SharedPreferences 中，并清空 EditText



图 9.5　单击“登录”按钮之后，取出数据并显示在 EditText 中

- 第一类是两个文本框：一个显示“用户名”，另一个显示“密码”，用来提示用户要输入什么样的数据。
- 第二类是两个编辑框，分别用来给供户输入用户名数据和密码数据。
- 第三类是按钮：一个“登录”按钮和一个“注册”按钮。

程序大致流程是，先在编辑框中输入内容，然后单击“注册”按钮，这个时候程序会将数据保存到 SharedPreferences 中，并清空编辑框。接着单击“登录”按钮，程序会从 SharedPreferences 中取出之前的数据并显示在编辑框中。

1．界面设计

接下来分析代码，首先是 xml 配置文件信息，如表 9-1 所示。

表 9-1　xml配置文件信息

控　　件	ID	重 要 参 数
“账号”文本框	@+id/name	android:text="账号"
“密码”文本框	@+id/password	android:text="密码"
“账号”编辑框	@+id/name_in	android:textSize="18sp"
“密码”编辑框	@+id/pass_in	android:textSize="18sp"
“登录”按钮	@+id/login	android:text="登录"
“注册”按钮	@+id/reg	android:text="注册"

2．Java代码整体设计

接下来是 Java 部分，首先是整体设计。在整体设计中，我们首先将在 xml 文件中定义的一些组件实例化，接着分别对登录和注册按钮设置单击的监听，最后在事件响应中实现内容的存储和读取功能，代码如下：

```
package com.wes.Chapter8;

import android.app.Activity;
import...                                              //省略部分导入的包
import android.content.SharedPreferences;      //导入 SharedPreferences 包
import android.content.SharedPreferences.Editor;
                                        //导入 SharedPreferences 编辑器包
public class SharedPreferencesTest extends Activity {
   /** Called when the activity is first created. */

    String name ;                       //声明变量 name 用来保存取出来的名字
    String pass ;                       //声明变量 pass 用来保存取出来的密码
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      Button loginBtn = (Button) findViewById(R.id.button0);
                                            //实例化登录按钮
      Button regBtn = (Button) findViewById(R.id.button1);
                                            //实例化注册按钮
      final EditText et1 = (EditText) findViewById(R.id.name_in);
```

```
                                                        //实例化用户名编辑框
        final EditText et2 = (EditText) findViewById(R.id.pass in);
                                                        //实例化密码编辑框

        regBtn.setOnClickListener(new OnClickListener()
                                                        //设置注册按钮的监听事件
        {
            @Override
            public void onClick(View arg0)  //这里实现保存功能
            {
            }
        });

        loginBtn.setOnClickListener(new OnClickListener()
                                                        //设置登录按钮的监听事件
        {
            @Override
            public void onClick(View v)     //这里实现读取功能
            {
            }
        });
    }
}
```

3．保存功能的实现

在注册按钮的事件响应中实现用户名和密码的保存功能，实现的步骤如下：

```
public void onClick(View arg0)
{
    name  = et1.getText().toString();           //取出用户名编辑框中的内容
    pass  = et2.getText().toString();           //取出密码编辑框中的内容
    //获得 SharedPreferences
    SharedPreferences sp = getSharedPreferences("Content", Context.MODE_
    PRIVATE);
    Editor editor = sp.edit();                  //获得编辑器
    editor.putString("name", name);             //保存用户名信息
    editor.putString("pass", pass);             //保存密码信息
    editor.commit();                            //提交修改
}
```

4．读取功能的实现

在登录按钮的响应事件中读取保存在 SharedPreferences 中的内容，并将其显示在编辑框中：

```
public void onClick(View arg0)
{
    //获得 SharedPreferences 对象
SharedPreferences sp = getSharedPreferences("Content", Context.
MODE_PRIVATE);
    String name = sp.getString("name", "");    //获得保存的用户名
    String pass = sp.getString("pass", "");    //获得保存的密码
    et1.setText(name);                          //显示用户名
    et2.setText(pass);                          //显示密码

}
```

首先获取 SharedPreferences 对象，然后得到名为 String 的 Key 所对应的 Value，一目了然。最后使用 setText()方法把它显示在 TextView 上。好了，关于 SharedPreferences 我们就先讲到这里，相信读者朋友应该学会使用 SharedPreferences 来存储数据了吧，那还在等什么？赶紧行动吧，试试看，你可以的！

9.2　使用文件存储

使用 SharedPreferences 保存数据固然是简单又方便，但很多时候用户使用它并不能很好地解决问题，这个时候使用文件存储会是一个很好的选择。那么，在 Android 中又是怎样使用文件来进行存储呢？文件保存在哪里呢？不要急，这些问题的答案你会在本节中一一找到。

9.2.1　文件保存概述

1．文件保存在哪里

和上一节一样，在开始学习使用文件存储之前，必须先了解它。而了解它的第一步就是：我们创建的文件到底保存在哪里呢？事实上，文件保存的路径与 SharedPreferences 的保存路径差不多，位于/data/data/<package name>/files 下，如图 9.6 所示。



图 9.6　文件保存路径

至于怎样找到这个<package name>，在 9.1 节中已经有过叙述，这里就不再赘言。我们导出文件会发现这个文件和普通的 txt 文件没有区别，这里就不再截图了。当然如果需要你也可以指定文件的后缀名为其他格式，如 tmp 等。

2．文件操作的一些方法

知道文件保存的位置和形式后，接下来需要知道操作文件的一些重要的方法，如表 9-2 所示。

表 9-2　文件操作的一些方法

操作文件的重要方法	含　义
openFileInput()	打开应用程序文件以便读取
openFileOutput()	创建应用程序文件以便写入
deleteFile()	通过名称删除文件
fileList()	获得所有位于/data/data/<package name>/files下的文件列表
getFileDir()	获得/data/data/<package name>/files子目录对象
getCacheDir()	获得/data/data/<package name>/cache子目录对象
getDir()	根据名称创建或获取一个子目录

学习到这里，我们对文件存储的一些准备工作已经就绪，上文列举的一些方法现在有些不明白没有关系，接下来将了解它们并学会使用它们。

9.2.2　在程序默认位置创建和写入文件

我们知道在 Java 中通过使用流来读写文件，要创建一个文件首先要建立一个输出流。至于流的概念因为超出本书的范畴，笔者这里就不展开去讲。在 Android 中同样如此，我们依靠 openFileOutput()来获得一个输出流。它将在上一小节中展示的位置创建一个文件，使用一个流要 3 个步骤。

1. 获得一个输出流对象

获得一个输出流对象以进行文件操作，使用 openFileOutput()方法可以很方便地获得，如下所示：

```
openFileOutput("myFile.txt",Context.MODE_PRIVATE);
```

这里需要两个参数，第一个参数是你需要创建的文件名，第二个参数是模式。至于各个模式的使用及其意义我们在上一节已经讨论，如有疑问请回顾上一节。

那么到这里很多读者会产生疑问，Java 中文件输出流都要指定文件路径的啊。不错，这就是 Android 提供的这个方法的方便之处了。使用 openFileOutput()方法不需要指定路径，系统会使用默认路径，也就是/data/data/<package name>/files 来保存你的文件。如果你在第一个参数中加入了路径，那么很不幸，程序会出现异常，所以该偷懒的时候就偷懒吧。

2. 向流中写入数据

获得了输出流之后，我们需要向流中添加我们需要添加的信息了，同样非常方便，使用 write()方法就可以了：

```
write(data.getBytes());
```

这里只有一个参数，就是 data.getBytes()，为什么呢？因为我们得到的 FileOutputStream 属于字节流，它只能按字节写入，也就是每次只写入一个字节。所以，String 型对象不能直接写入，必须将其转换为 Byte 型。当然，你也可以使用 OutputStreamWriter 将其转换为字符流再进行操作。I/O 相关的内容，本书不会展开去讲，读者如果有兴趣可以深入学习。

3. 关闭流

当数据写入完毕后，使用 close()方法可以关闭输出流，方法如下：

```
fos.close();
```

在每次使用完流之后，我们要记得及时将其关闭。要学会在平时养成良好的编程习惯，一点一滴的积累会造就你卓越的编程技巧。

9.2.3　在默认位置读取文件

与上面类似，Android 提供了读取文件的简便方法，同样需要 3 个步骤。

1. 创建输入流

学会了创建输出流后再学习创建输入流就变得非常简单了，当然这里有一些不同需要注意，下文会有解释，方法如下：

```
FileInputStream fis;
InputStreamReader isr;
BufferedReader  br;
fis = openFileInput("myFile");
isr = new InputStreamReader(fis);
br  = new BufferedReader(isr);
```

这里为了将读取出来的内容保存在 String 中，我们对 FileInputStream 进行了包装，其实只有一个目的：获得可以直接读取 String 的输入流。

首先，得到了一个输入字节流，参数是文件名：

```
fis = openFileInput("myFile");
```

接着，将其转换为字符流，这样可以一个字符一个字符地读取以便显示中文：

```
isr = new InputStreamReader(fis);
```

最后，我们又将其包装为缓冲流，这样可以一段一段地读取，减少读写的次数，保护硬盘：

```
br  = new BufferedReader(isr);
```

那么到此为止，我们得到了一个理想的输入流。

2. 读取数据

从流中获得数据同样非常方便，这里没有使用 read()方法而使用了 readLine()方法，原因下文也会给出，代码如下：

```
String s = null;
S = (br.readLine());
```

这里使用了 readLine()方法，这也是我们将 FileInputStream 包装的原因。因为这个方法是一行一行地读取，使用非常方便。将数据读取出来之后可以将其保存在内存中以便操作。

3. 关闭输入流

这里记得每个流都要关闭哦：

```
fis.close();
isr.close();
br.close();
```

同之前一样，每次使用完流之后，我们要记得及时将其关闭，以保证程序的正常稳定运行。

9.2.4　通过实例学习文件存储

接下来的例子演示了怎样通过文件保存和读取数据。为了方便，我们依然使用 Shared-

Preferences 实例中的布局，实现同样的功能，只是修改其中的 onClick()方法，将保存和读取数据的方法改为通过文件完成。效果图就不再贴出，大家通过代码可以调试一下。

1．整体设计

同样的，在整体设计中首先实例化 xml 文件中定义的一些组件，接着分别对登录和注册按钮设置监听事件：

```
package com.wes.fileIO;

import Java.io.BufferedReader;                          //导入缓冲字符输入流包
import Java.io.FileInputStream;                         //导入文件输入字节流包
import Java.io.FileOutputStream;                        //导入文件输出字节流包
import Java.io.InputStreamReader;                       //导入字符输入流包
import ...                                              //省略部分包的导入

public class mainActivity extends Activity {
   /** Called when the activity is first created. */

    String PASS = "myPassWord.txt";                     //声明文件名
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);

       Button loginBtn = (Button) findViewById(R.id.button0);
                                                        //实例化登录按钮
       Button regBtn = (Button) findViewById(R.id.button1);
                                                        //实例化注册按钮
       final EditText et2 = (EditText) findViewById(R.id.pass in);
                                                        //实例化编辑框

       regBtn.setOnClickListener(new OnClickListener()
                                                        //设置注册按钮单击事件
       {
            @Override
            public void onClick(View arg0)              //这里实现保存功能
            {}
            }
       });

       loginBtn.setOnClickListener(new OnClickListener()
                                                        //设置登录按钮单击事件
       {
            @Override
            public void onClick(View arg0)              //这里实现读取功能
            {}
       });
   }
}
```

2．保存功能的实现

在注册按钮的响应事件中实现保存功能，实现的过程其实并不复杂，注意要使用 try{…}catch(){…}语句包裹文件流操作，以捕获异常：

```
public void onClick(View arg0)
            {
                String pass = et2.getText().toString();
                                             //取得密码编辑框中的内容
                try
                { //获得 PASS 文件的文件字节输出流
                    FileOutputStream fos = openFileOutput(PASS,Context.
                    MODE_PRIVATE);
                    fos.write(pass.getBytes());  //写入内容
                    fos.close();                 //关闭输出流
                    et2.setText("");             //清空编辑框
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
```

3．读取功能实现

在登录按钮的响应事件中实现读取功能，同样使用 try{…}catch(){…}语句包裹文件流操作，以捕获异常：

```
public void onClick(View arg0)
            {
                FileInputStream fis;
                InputStreamReader isr;
                BufferedReader  br;
                try
                {
                    fis = openFileInput(PASS);      //获得文件字节输入流
                    isr = new InputStreamReader(fis);  //包装为字符流
                    br  = new BufferedReader(isr);
                                            //再包装为带缓冲的字符流
                    String s = "";          //声明变量 s 保存读取每行字符
                    StringBuffer sb = new StringBuffer();
                                              //声明变量 sb 保存所有内容
                while (( s = br.readLine())!= null)
                                              //按行读取流中的数据
                        {
                            sb.append(s + "\n");//将每行字符“拼接”起来
                        };
                    et2.setText(sb);          //将读取的内容显示
                    fis.close();              //关闭 fis 流
                    isr.close();              //关闭 isr 流
                    br.close();               //关闭 br 流
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
```

相信这段代码读者理解起来应该不会有太大的困难，那么，还在犹豫什么？赶快动手尝试一下吧。

9.3　使用 SQLite 数据库

本节将讲解使用 Android 自带的关系型数据库——SQLite，它是一个基于文件的轻量级数据库，为嵌入式设备量身打造。每个应用程序创建的数据库都是私有的，但是 ContentProvider 可以把数据共享给其他的应用程序。

本节中读者将学习创建和删除数据库、创建和删除表，以及插入记录、更新记录、删除记录和查询记录等操作。

9.3.1　创建和删除数据库

1. 怎样创建数据库

创建数据库有多种方法，最简单的无疑是使用 Context 的 openOrCreateDatabase()方法了。当然还有更加强大的方法，比如通过 SQLiteOpenHelper 类更有效地管理它。本节我们首先讲解第一种简单的方法，SQLiteOpenHelper 类的使用将在之后的内容中讲解。

使用 openOrCreateDatabase()方法创建数据库的语法格式如下：

```
ContextWrapper.openOrCreateDatabase(String name, int mode, Cursor- Factory
factory)
```

这里需要 3 个参数：

- ❑ String name：数据库的名字，每个数据库的名字都是独有的，注意要以“.db”为后缀名。
- ❑ Int mode：数据库的模式，一般设置为 SQLiteDatabase.CREATE_IF_NECESSARY。
- ❑ CursorFactory factory：工厂类的对象，在执行查询时通过该工厂创建一个 Cursor 类。Cursor 类的使用会在本节之后的内容中讲解。这里我们不需要它，可以将之设为 null。

所以创建一个名为 my_database.db 的数据库语法格式如下：

```
SQLiteDatabase  db  =  openOrCreateDatabase("my database.db",  SQLiteDat-
abase. CREATE_IF_NECESSARY, null);
```

2. 数据库文件被保存在哪里

前文讲过，SQLite 数据库是基于文件的关系型数据库，那么我们创建的数据库又被保存在哪里呢？事实上，与 SharedPreferences 类似，数据库文件被保存在如下的目录下：

```
/data/data/package name/databases
```

通过 DDMS 查看我们刚才创建的数据库文件，其位置如图 9.7 所示。

图 9.6 中的 my_database.db 就是刚才创建的数据库文件了。

3. 设置数据库

创建完数据库后，为了更安全而有效地使用它，我们还需要对它进行一定的设置，主

要的方法有 3 个，分别是：

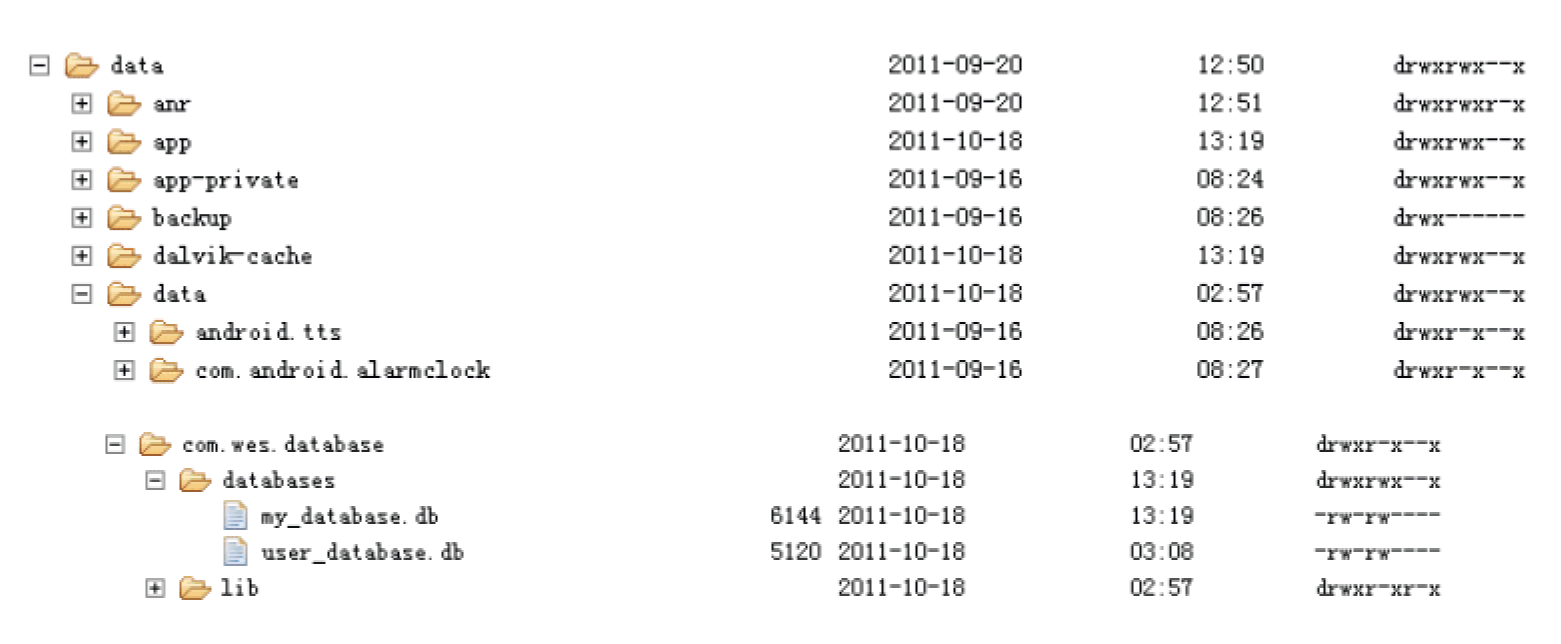


图 9.7 数据库文件保存位置

（1）设置本地化：

```
db.setLocale(Locale.getDefault());
```

该方法的参数我们设置为默认，当然也可以设置为对应的区域，如设置为 Locale.CHINA。

（2）设置线程安全锁：

```
db.setLockingEnabled(true);
```

在平常的使用中经常会有不同的线程在同时操作数据库，这个时候就需要使用线程安全锁来保证数据库在一个时间上只被一个线程使用。

（3）设置版本：

```
db.setVersion(1);
```

数据库可能会经常更新，为了更有效地管理，我们为每个数据库设置了版本。

4. 关闭数据库

当我们不再需要使用数据库时可以考虑将其关闭，关闭的方法非常简单，只需要调用方法：

```
db.close();
```

5. 删除数据库

有些时候基于某种需求我们需要将数据库彻底删除，方法同样非常简单：

```
Context.deleteDatabase();
```

注意：数据库的删除是永久且不可恢复的，所以执行该方法时请务必慎重，不要将删除数据库与下小节将要讨论的删除表混淆。

9.3.2 创建和删除表

1. 创建表

创建了数据库之后，我们接下来的工作就是在数据库中创建表了，只有拥有了表，数

据库才有意义。创建表的方法是执行相应的 SQL 语句。例如，要创建一个名为 userInfo_brief 的表，需要的 SQL 语句为：

```
CREATE  TABLE userInfo_brief (
id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT,
password TEXT);
```

对应的 Java 代码如下：

```
String TABLENAME_1 = "userInfo_brief";
String ID = "id";
String NAME = "name";
String PASSWORD = "password";
String sql = "CREATE TABLE " +
           TABLENAME_1 +"(" +
           ID + " INTEGER PRIMARY KEY  AUTOINCREMENT," +
           NAME + " TEXT," +
           PASSWORD + " TEXT);";
           db.execSQL(sql);
```

如上所示，表中有 3 列，第一列是 id，之后的 3 个修饰符的意义是：整数型、主键、自动增长。第二列是账户（name），类型为 TEXT，文本型。第三列是密码（password），类型同样为 TEXT，文本型。

注意加粗部分的代码，编写完 SQL 语句后需要执行它，方法就是：

```
SQLiteDatabase.execSQL(String sql);
```

该方法常被用于执行非查询操作，如创建表、删除表、更新表等。

2. 删除表

相应的，如果要删除表 userInfo_brief ，其 SQL 语句为：

```
DROP TABLE userInfo_brief;
```

与其对应的 Java 代码为：

```
Sring sql = "DROP TABLE "+ TABLENAME 1 + ";";
db.execSQL(sql)
```

9.3.3　操作记录

操作记录大致包括插入操作、更新操作以及删除操作。SQLiteDatabase 类提供了 3 个简单的方法来完成对应的操作，它们分别是：

```
SQLiteDatabase.insert()
SQLiteDatabase.update()
SQLiteDatabase.delete()
```

1. 插入记录

插入记录时，我们需要使用方法：

```
SQLiteDatabase.insert(String table, String nullColumnHack, ContentValues
values)
```

这里有 3 个参数：

- 表名：需要操作的表名。
- 空列：数据库不允许插入一条完全是空的记录，所以如果初始值是空的，那么该列会被标志为 NULL。
- 内容值：　需要插入到表中的数据，数据类型为 ContentValues。

那么 ContentValues 如何使用呢？ContentValues 有些类似于 HashMap，同样是以键值对的形式保存数据。使用它需要两个步骤：

（1）获得 ContentValues 对象：

```
ContentValues values = new ContentValues();
```

（2）以键值对的形式保存数据，“键”为列名，“值”为真正需插入到表中的数据：

```
values.put(NAME, "wesley");
```

向不同的列中插入数据可以多次执行 put()方法。假设要向 userInfo_brief 表中插入一条记录，语法格式为：

```
ContentValues values = new ContentValues();
        values.put(NAME, "wesley");
        values.put(PASSWORD, "123wes");
        db.insert(TABLENAME_1, null, values);
```

除了使用 insert()方法外，开发者还可以使用 insertOrThrow()方法完成插入操作，不同的是，该方法在插入操作失败时会抛出 SQL 异常，有利于调试程序。

2．更新记录

更新记录时，我们需要使用方法：

```
SQLiteDatabase.update(String table, ContentValues values, String where-
Clause, String[] whereArgs)
```

这里有 4 个参数：

- 表名：需要操作的表名。
- 内容值：要更新的数据。
- Where 子句，“？ ”表示子句参数，为 null 时表示更新所有的记录。
- 子句参数组，子句中的每个字符串会替代 Where 子句中相应的“？ ”。只有在第三个参数非 null 时，该参数才有效。

例如，要更新 userInfo_brief 表中 name 为 wesley 的密码，相应的代码为：

```
ContentValues values = new ContentValues();
        values.put(PASSWORD, "123456");
        db.update(TABLENAME_1, values, NAME+"=?", new String[]{"wesley"});
```

这里的 ContentValues，我们只保存了一个键值对，因为我们只需修改这一个列，其他不需要的字段则无需保存。

3. 删除记录

删除记录时，我们需要使用方法：

```
SQLiteDatabase.delete(String  table,  String  whereClause,  String[]
whereArgs)
```

这里有 3 个参数：

- 表名：需要操作的表名。
- Where 子句，“？ ”表示子句参数，为 null 时表示删除所有的记录。
- 子句参数组，子句中的每个字符串会替代 Where 子句中相应的“?”。同样的，只有在第二个参数非 null 时，该参数才有效。

例如，要删除 userInfo_brief 表中 name 为 wesley 的记录，代码为：

```
db.delete(TABLENAME_1, NAME+"=?", new String[]{"wesley"});
```

如果要删除表中的所有数据则执行：

```
db.delete(TABLENAME_1,null,null);
```

4. 同时执行多个操作

学习了 3 种基本操作之后我们可以尝试着将其组合起来。例如，我们需要执行以下操作：

（1）删除一条记录。

（2）插入另一条记录。

（3）更新插入的记录。

要求：如果插入操作或更新操作失败，则不改变原有记录，也就是不执行删除操作。以上 3 个步骤我们已经可以通过之前的学习写出相应的代码：

```
SQLiteDatabase.delete(String  table,  String  whereClause,  String[]
whereArgs)
SQLiteDatabase.insert(String table, String nullColumnHack, ContentValues
values)
SQLiteDatabase.update(String  table,  ContentValues  values,  String
whereClause, String[] whereArgs)
```

可是，假设在第二步插入记录时发生了错误，这个时候删除操作已经被执行，这种情况显然是我们所不希望看见的。那么怎样才能完成要求呢？这时我们就引入了事务（Transaction）的概念。

事务的作用是：当你希望执行一系列操作时，你希望这些操作要么都被执行要么都不被执行。使用事务后，如果某个操作失败，我们可以处理错误，也可以恢复操作。如果操作成功则可以提交操作。使用事务的方法为：

```
//事务的开始必须调用 beginTransaction()方法，并以 endTransaction()方法结束
SQLiteDatabase.beginTransaction()
try
          {
           //执行你希望执行的操作，如插入，更新，查询等
```

```
insert();
            update();
            select();
//顺利执行完操作后，需要调用以下方法使设置生效
            SQLiteDatabase.setTransactionSuccessful();
           }
           catch(Exception e)
           {
            //这里可以执行一些当操作发生错误时你希望进行的操作，如提示用户操作失败，
              请检查等
            e.printStackTrace();
           }
           finally
           {
//最后需要结束事务
SQLiteDatabase.endTransaction();
           }
```

注意所有的操作必须被包含在 try/catch 语句块中，要使操作生效必须调用 SQLiteDatabase.setTransactionSuccessful()方法，如果没有调用该方法，则之前进行的所有操作都将被回滚，也就回到了没有进行任何操作时的状态。同时，完成操作后必须调用 endTransaction()方法。当然，如果你需要事务也是可以被嵌套的。

9.3.4　查询记录

我们为什么要使用数据库来保存数据？如果仅仅是为了保存数据，有很多种方法也许比数据库更好。数据库之所以强大就是因为我们可以通过一系列的查询语句，来得到我们希望得到的数据记录，从而避免了繁琐的查找工作，提高了效率。

1．使用query()查询方法

SQLite 数据库提供了一些方法可以使我们很方便地执行查询，而避免了写一些繁琐的 Select 查询语句，当然，你也不必担心，因为它提供的方法其实与 SELECT 语句一脉相承。其语法格式如下：

```
SQLiteDatabase.query(String table, String[] columns, String selection,
String[] selectionArgs, String groupBy, String having, String orderBy)
```

该方法需要 7 个参数，看上去是不是很可怕？不要担心，很多时候一些参数我们根本不需要使用，这个时候使用 null 替代就可以了。那么接下来，让我们看一下到底这些长长的参数到底各自有着怎样的作用呢？

（1）String table：该参数为表名，也就是你希望要进行查询的表。

（2）String[] columns：列名，也就是你希望查询的一组属性。

（3）String selection：选择条件，也就是 SELECT 语句中的 WHERE。

（4）String selectionArgs：选择条件的参数。例如，在第三个参数中填写了“id=?”，那么第四个参数则替代第三个参数中的?

（5）String groupBy：分组，顾名思义，作用与 SELECT 语句中的 GROUPBY 相同。

（6）String having：条件，在进行分组之后继续筛选数据，通常包含有聚合函数，其作用与 Where 相同，不过 Where 不能和聚合函数一起使用。例如，HAVING MIN(column)> 5

就是对分组之后的数据再次筛选，选出最小的 Cursor 大于 5 的记录。其中，MIN(column)被称为聚合函数。

（7）String orderBy：排序，按照某一项升序或者降序排列，ASC 为升序，DESC 为降序。

2．使用Cursor对象保存查询结果

那么查询之后的结果呢？结果是以怎样的形式返回的呢？事实上，查询的结果一般以 Cursor 对象的形式保存并返回给用户。Cursor 对象就好比是一个文件指针，使用它可以很方便地对结果进行遍历。返回的 Cursor 对象我们可以将之想象为一张表，每条记录构成一行，每个查询的属性构成一列。注意这里的列数由第二个参数中 String 数组的个数决定，而非由数据库中表的具体列数决定。当然如果第二个参数是 null 那么结果就是表的所有列。

Cursor 对象只能暂时地保存数据，如果只需要完成一些简单的操作可以快速地执行，执行完后关闭它。

接下来学习一些经常使用的 Cursor 对象的方法：

（1）Cursor.getCount()，获得 Cursor 对象中记录条数，可以理解为有几行。

（2）Cursor.getColumnCount()，获得 Cursor 对象中记录的属性个数，可以理解为有几列。

（3）Cursor.moveToFirst()，将 Cursor 对象的指针指向第一条记录。

（4）Cursor.moveToNext()，将 Cursor 对象的指针指向下一条。

（5）Cursor.isAfterLast()，判断 Cursor 对象的指针是否指向最后一条记录。

（6）Cursor.close()，关闭 Cursor 对象。

（7）Cursor.deactivate()，取消激活状态。

（8）Cursor.requery()，重新查询刷新数据。

学习了这些方法之后，我们还需要进行一个认知：我们必须对 Cursor 随时保持关注，并很好地管理它。例如，在应用程序关闭或者暂停时，也就是在 onPause()或者 onStop()方法中，我们要使用 deactivate()方法取消激活；当程序恢复时，也就是在 onResume()方法中，我们需要使用 requery()方法刷新数据。最后不再使用 Cursor 或者退出程序时要关闭 Cursor，也就是在 onDestroy()方法中，调用 close()方法以释放资源。

也许你会想，使用 Cursor 也太麻烦了吧！没有关系，就像很多人不会使用单反相机，厂商推出了傻瓜式相机一样。Android 为我们提供了一个方便的方法管理 Cursor 对象：

```
Activity.startManagingCursor(Cursor c)
```

将那些繁琐的工作都交给系统来管理吧，激活、取消激活、关闭对象等都不需我们动手啦。当然，如果你有需要，随时可以“接管”Cursor 的管理。方法为：

```
Activity.stopManagingCursor(Cursor c)
```

使用之前介绍的 Cursor.moveToFirst()、Cursor.moveToNext()、Cursor.isAfterLast()3 种方法可以很好地完成对结果的迭代。

例如，遍历一个名为 c 的 Cursor 对象，我们可以使用如下代码段：

```
c.moveToFirst();                        //将指针指向第一条记录
        while(!c.isAfterLast())         //判断是否为最后一条数据
```

```
        {
            String result = "";
            for(int i = 0; i < c.getColumnCount(); i++)
            {
                result = result.concat(c.getString(i)+" ");
                                                //得到所有的列数据
            }
            ...                                 //对结果进行一些操作
        c.moveToNext();                         //指向下一条数据
        }
```

3．执行查询操作

接下来我们就从最简单的查询语句开始。

【例 1】我们都知道，要查询一张表中的所有数据的 SELECT 语句为：

```
SELECT  *  FROM  TABLENAME
```

那么，使用 Android 为我们提供的查询方法为：

```
Cursor  c = database.query(TABLENAME,null,null,null,null,null,null);
```

通过该方法可以得到名为 TABLENAME 表中的所有数据。

【例 2】如果希望一次查询只返回一条记录，我们可以使用的 SELECT 语句为：

```
SELECT * FROM  TABLENAME  WHERE id=1
```

这条语句的意思是查询 id 为 1 的记录的所有信息。相应的查询方法为：

```
Cursor  c = database.query(TABLENAME,null, "id=?",new String[]{"1"},,
null,null,null);
```

【例 3】如果希望一次查询表中的若干属性，SELECT 语句可以为：

```
SELECT name，password  FROM  TABLENAME  WHERE id=1
```

该语句查询了表中 id 为 1 的记录的 name 和 password 属性。相应的查询方法为：

```
Cursor  c =  database.query（TABLENAME,new String[]{"name","password"},
"id=?",new String[]{"1"},,null,null,null）;
```

【例 4】如果希望一次查询表中的若干属性，并按照一定的属性升序排列，SELECT 语句可以为：

```
SELECT name，password，age  FROM  TABLENAME  WHERE sex=男 ORDERBY age ASC
```

该语句查询了表中性别为男的记录的 name、password 和 age 属性。结果按照年龄升序排列。相应的查询方法为：

```
Cursor  c = database.query(TABLENAME,new String[]{"name","password",
"age"}, "sex=?",new String[]{"男"},,null,null, "age ASC");
```

【例 5】如果希望一次查询表中的若干属性，并按男女分组，同时按照某属性降序排列，SELECT 语句可以为：

```
SELECT name，password，age  FROM  TABLENAME  WHERE area="江苏" GROUPBY sex
ORDERBY age ASC
```

该语句查询了表中所在地为江苏的记录的 name、password 和 age 属性。结果按照年龄升序排列。相应的查询方法为：

```
Cursor  c = database.query（TABLENAME,new String[]{"name","password",
"age"}, "area=?",new String[]{"江苏"},,"sex",null, "age ASC"）;
```

4. 使用SQLiteQueryBuilder

之前我们学习的都是一些简单的单表查询，如果我们需要执行多表查询时怎么办呢？这个时候 SQLiteQueryBuilder 类就可以大显身手了。例如，现在拥有两张表，分别是 user_brief 表和 user_detail 表。brief 表中只是保存了用户名和密码，detail 表中则存储了用户名和其他具体信息，如年龄、国籍、爱好等。接下来我们需要执行多表查询：

```
SELECT  user brief.name,
            User brief.password,
            User detail.age,
            User detail.sex
FROM        user brief, user detail
WHERE   user_brief.name = user_detail.name AND user_brief.name = "wes"
ORDERBY     age ASC
```

这个时候我们可以使用 SQLiteQueryBuilder 类来帮助我们完成查询。要使用 SQLiteQueryBuilder 需要以下步骤：

（1）获得 SQLiteQueryBuilder 对象：

```
SQLiteQueryBuilder builder = new SQLiteQueryBuilder();
```

（2）设置需要查询的表，各个表之间以“，”隔开：

```
builder.setTables(TABLENAME_1+","+TABLENAME_2);
```

（3）设置关联属性，表与属性之间以“.”隔开，两属性之间以“=”连接：

```
builder.appendWhere(TABLENAME_1+"."+NAME +"="+ TABLENAME_2+"."+NAME);
```

（4）开始查询：

```
cursor = builder.query(SQLiteDatabase db, String[] projectionIn, String
selection, String[] selectionArgs, String groupBy, String having, String
sortOrder)
```

这里同样需要 7 个参数：

（1）SQLiteDatabase db：数据库名，需要进行操作的数据库。

（2）String[] projectionIn：类似于普通查询的 columns，意义是希望查询的属性。

（3）String selection：选择条件，也就是 SELECT 语句中的 WHERE。

（4）String selectionArgs：选择条件的参数，用来替代第三个参数中的？

（5）String groupBy：分组，也就是 SELECT 语句中的 GROUPBY。

（6）String having：条件，作用在前文中已经解释过。

（7）String orderBy：排序条件。

了解了以上的步骤后，接下来我们就可以完成前文写的 SELECT 语句了，其代码片段如下：

```
String TABLENAME_1 = "user_brief";
String TABLENAME_2 = "user_detail";
String ID = "id";
String NAME = "name";
String SEX = "sex";
String AGE = "age";
String PASSWORD = "password";
SQLiteQueryBuilder builder = new SQLiteQueryBuilder();
builder.setTables(TABLENAME_1+","+TABLENAME_2);    //设置表
builder.appendWhere(TABLENAME_1+"."+NAME +"="+ TABLENAME_2+"."+NAME);
String[] colums = {TABLENAME_1+"."+NAME+"," +
                   TABLENAME_1+"."+PASSWORD+","+
                   TABLENAME_2+"."+SEX+","+
                   TABLENAME_2+"."+AGE};
cursor = builder.query(db, colums, TABLENAME_1+"."+NAME+"= ? ", new
String[]{"wes"}, null, null, "age ASC");
```

数据库的查询同样是一门很大的课题，本书限于篇幅只是介绍了一些简单的查询方法，更多的方法还需要读者自行钻研。

9.3.5　使用数据库帮助类

通过之前 4 个小节的讲解，我们已经基本上可以完成一些简单的数据库操作了。但是，在实际的应用开发中，程序员并不是在程序运行时创建数据库，然后操作它，最后程序退出时删除它。我们使用数据库的目的往往是持久地保持数据以备使用。为了能更好地管理数据库，Android SDK 为我们提供了数据库的一个帮助类——SQLiteOpenHelper。

要使用 SQLiteOpenHelper 类需要如下步骤。

1．继承SQLiteOpenHelper类

要使用该帮助类，首先我们要继承它，并重写 onCreate()方法、onOpen()方法以及 onUpgrate()方法，这一点与 Activity 比较类似。在数据库被创建时，系统会调用 onCreate()方法，一般在该方法中会完成表的创建；在数据库更新时系统会调用 onUpgrade()方法，在这个方法里你可以完成一些在数据库更新时希望进行的一些工作，如提醒用户数据库版本发生改变等。以下给出了一个简单的帮助类的实现代码：

```
package com.wes.database;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHelper  extends SQLiteOpenHelper
{
    String DATABASENAME ="my_database.db";
    String TABLENAME_1 = "userInfo_brief";
    String TABLENAME_2 = "userInfo_detail";
    String ID = "id";
    String NAME = "name";
    String SEX = "sex";
    String OLD = "old";
    String PASSWORD = "password";
```

```
    public DatabaseHelper(Context context, String name, CursorFactory
factory, int version)
    {
        super(context, name, factory, version);
        // 构造函数
    }

    @Override
    public void onCreate(SQLiteDatabase db)
    {
        //一般在这里完成表的创建，当然你也可以完成一些其他你希望完成的步骤
        //创建 userInfo_brief 表
String sql = "CREATE TABLE " +
        TABLENAME_1 +"(" +
        ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
        NAME + " TEXT," +
        PASSWORD + " TEXT);";
        db.execSQL(sql);

        //创建 userInfo_detail 表
        String sql2 = "CREATE TABLE " +
        TABLENAME_2 +"(" +
        ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
        NAME + " TEXT," +
        SEX + " TEXT," +
        OLD + " TEXT);";
        db.execSQL(sql2);
    }
    @Override
    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2)
    {
        // 做一些你希望在更新数据库时进行的操作
    }

    @Override
    public void onOpen(SQLiteDatabase arg0)
    {
        super.onOpen(db);
//完成一些在打开数据库时希望完成的工作
    }
}
```

不过要注意的是，与 Activity 调用不同的是，并非每次得到数据库的实例，系统都会调用 onCreate()方法，而是只有在数据库被第一次创建时该方法才会被调用。onOpen()方法则是在每次打开数据库时都会被调用。

2．得到帮助类的对象

通过帮助类的构造方法可以顺利地得到帮助类的对象：

```
DatabaseHelper helper = new DatabaseHelper(Context context, String name,
CursorFactory factory, int version)
```

注意这里的构造方法有 4 个参数：

（1）Context context：上下文关系，这里不再赘述。

（2）String name：数据库的名字，前文提到 SQLite 数据库是以数据库的名字为标示来

打开或操作的。

（3）CursorFactory factory：Cursor 工厂，当你查询时自动生成一个 Cursor 对象，一般该参数我们设置为空。

（4）int version：版本，设置数据库的版本以便管理和更新。

3．获得数据库

顺利地获得数据库帮助类的对象后，无论在什么时候，我们都可以很方便地得到一个可读或者可写数据库，得到可写数据库的具体语法格式如下：

```
SQLiteOpenHelper.getWritableDatabase()
```

或者可以通过以下方法得到可读数据库：

```
SQLiteOpenHelper.getReadableDatabase()
```

通过以上的 3 个步骤我们可以更高效地使用数据库来帮助我们开发。本节所有的语法结构在示例 DatabaseDemo 中都有使用，读者可以通过观察程序源代码进一步了解和使用数据库。

9.4　实例——通过数据库验证登录

通过 9.3 节的学习，相信读者已经掌握了基本的使用数据库的方法。这一节我们将使用数据库来完成一个简单的登录应用。通过本实例的学习读者可以巩固数据库的建立、表的建立、插入记录、查询记录等一系列的数据库操作。

本实例由 4 部分组成：登录界面、注册界面、登录成功界面以及数据库部分。

9.4.1　整体设计

在开始编写界面之前，我们首先要设计好准备使用的数据库。本例的需求是：

（1）程序开始进入登录界面。

（2）在登录界面中输入用户名和密码，单击“登录”按钮，若成功则跳转到登录成功界面，登录失败提示用户“密码错误”；若没有用户名则单击“注册”按钮，跳转到注册页面。

（3）注册页面中，提示用户输入用户名、密码、年龄、爱好等选项。输入完成后单击“确定”按钮，跳转到登录成功界面，并将数据保存到数据库。

（4）登录成功界面从数据库提取该用户相关信息显示。

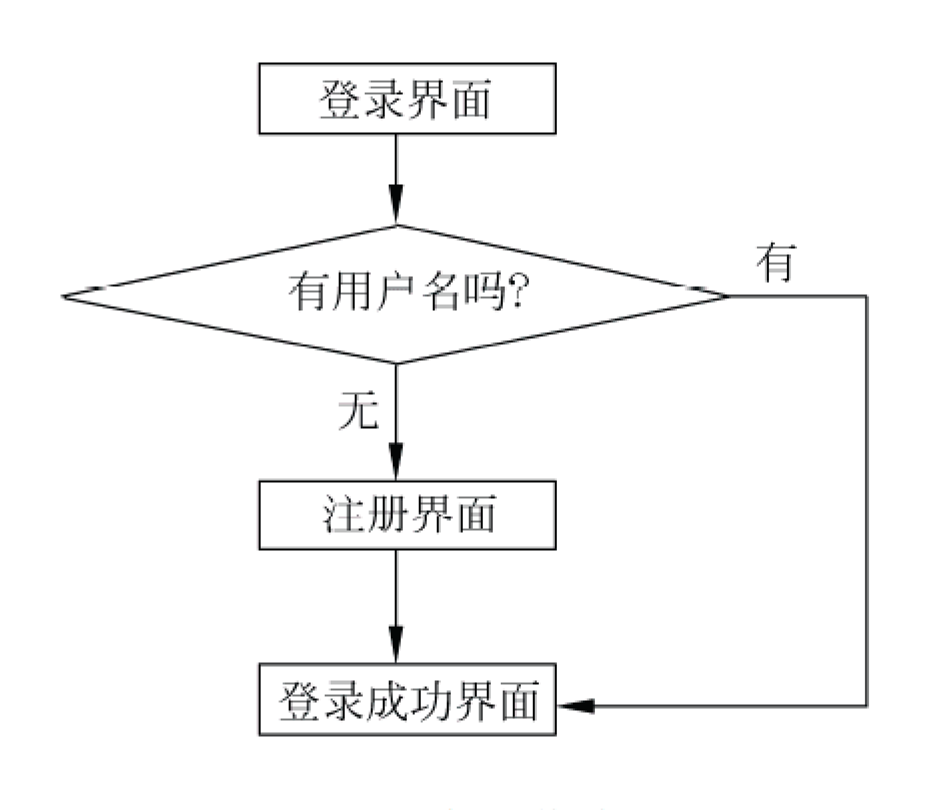


图 9.8　用户操作流程图

流程如图 9.8 所示。

9.4.2　数据库设计

依照 9.4.1 小节中的设计需要，我们只需设计一张表就可以完成。用户信息表的设计如表 9-3 所示。

表 9-3　用户信息表

属　　性	类　　型	含　　义	属　　性	类　　型	含　　义
Id	INTEGER	主键Id	Age	INTEGER	年龄
Name	text	名字	Sex	text	性别
Password	text	密码	Hobby	text	爱好

接下来完成 SQLiteOpenHelper 类的编写，在第一次创建数据库时新建用户信息表。DatabaseHelper 代码如下：

```
package com.wes.Chapter9;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHelper  extends SQLiteOpenHelper
{
    final static String DATABASENAME ="my database.db";
    final static int    VERSION          = 1;
    final static String TABLENAME   = "userInfo detail";
    final static String ID               = "id";
    final static String NAME             = "name";
    final static String SEX              = "sex";
    final static String AGE              = "age";
    final static String HOBBY            = "hobby";
    final static String PASSWORD         = "password";

    public DatabaseHelper(Context context)
    {
        super(context, DATABASENAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db)
    {
        String sql2 = "CREATE TABLE " +
        TABLENAME   + "(" +
        ID          + " INTEGER PRIMARY KEYAUTOINCREMENT," +
        NAME        + " TEXT," +
        PASSWORD    + " TEXT," +
        SEX     + " TEXT," +
        AGE     + " TEXT," +
        HOBBY       + " TEXT);";
        db.execSQL(sql2);
    }

    @Override
    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2)
    {
```

```
    }
}
```

注意：这里的构造函数，即加粗部分，为了在使用时更加方便，我们省略了部分构造函数的参数，只保留了上下文关系 context，其他的参数在本类中定义，这样在使用的时候更加简单，代码之间耦合更低。

9.4.3　登录界面设计

本例在登录界面中准备实现两个功能，首先是跳转到注册界面，这一功能与数据库无关。另一个功能是完成登录，直接跳转到登录成功界面，这部分就需要到数据库中查询该用户名注册的密码并与用户本次输入的密码验证。若成功才进行跳转，若不成功则提示用户密码错误，接下来进入具体的设计过程。

1．页面设计

登录界面需要使用的组件如表 9-4 所示。

表 9-4　登录界面组件表

类　　型	ID	含　　义	类　　型	ID	含　　义
TextView	Name	提示用户输入用户名	EditText	Pass_in	密码输入框
EditText	Name_in	用户名输入框	Button	Button0	登录按钮
TextView	Pass	提示用户输入密码	Button	Button1	注册按钮

将这些组件按照你希望的样子进行布局，完成你自己的个性登录界面吧。

2．整体设计

首先得到组件的操作对象，接着分别完成注册按钮的跳转功能和登录按钮的判断以及跳转功能：

```
package com.wes.Chapter9;

import ...                              //省略部分导入
import android.content.ContentValues;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.widget.Toast;

public class LoginActivity extends Activity {
    String name ;
    String pass ;
    String sex ;
    String age ;
    String hobby ;
    SQLiteDatabase db;                  //数据库对象

   @Override
```

```
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
         //获得组件的操作对象
        Button loginBtn = (Button) findViewById(R.id.button0);
        Button regBtn = (Button) findViewById(R.id.button1);
        final EditText et1 = (EditText) findViewById(R.id.name_in);
        final EditText et2 = (EditText) findViewById(R.id.pass_in);

        regBtn.setOnClickListener(new OnClickListener()
        {
             //完成注册按钮的功能
        });



        loginBtn.setOnClickListener(new OnClickListener()
        {
             //完成登录按钮功能
        });
    }
}
```

3．注册按钮功能设计

使用 Intent 完成从 LoginActivity 到 RegisterActivity 的跳转：

```
@Override
            public void onClick(View arg0)
            {
                Intent i = new Intent(LoginActivity.this,RegisterActivity.
                class);
                startActivity(i);             //开始跳转
            }
```

4．登录按钮功能设计

在完成按钮功能时需要分为 4 步：

（1）得到数据库操作对象。

（2）从输入框中分别获得用户名和密码。

（3）从数据库中提取该用户名的密码，若没有得到相关记录则提示该用户不存在。若提取成功则进入密码比对。

（4）若提取的密码与输入的密码相同则跳转到登录成功，否则提示“密码错误”。

登录按钮功能相关代码如下：

```
@Override
            public void onClick(View v)
            {
                //首先通过帮助类得到数据库操作对象
                DatabaseHelper helper = new DatabaseHelper(getBaseContext());
                db = helper.getReadableDatabase();
                //得到用户输入的信息
                 name = et1.getText().toString();
                pass = et2.getText().toString();
```

```
                //根据用户名查询数据库信息
                Cursor cursor = db.query(DatabaseHelper.TABLENAME,
                                     new String[]{DatabaseHelper.
                                     PASSWORD},
                                     DatabaseHelper.NAME + "=?",
                                     new String[]{name}, null, null,
                                     null);
                //若没有查询到相关信息则提示用户名不存在，不再继续操作
if (cursor.getCount() == 0 )
                {
Toast.makeText(getBaseContext(), "该用户名不存在！！", Toast.LENGTH LONG).
show();
                    return;
                }
                //若用户名存在，则继续操作
                cursor.moveToFirst();                    //指向第一条记录
                String password = cursor.getString(0); //取得密码
                //判断密码，若一样则进行跳转，否则提示密码错误
                if (password.equals(pass))
                {
                    Intent i = new Intent(LoginActivity.this,SucAct-
                    ivity.class);
                    i.putExtra("name", name);
                    startActivity(i);
                }
                else
                {
                    Toast.makeText(getBaseContext(), "密码错误！！", Toast.
                    LENGTH LONG).show();
                }
            }
```

9.4.4　注册界面设计

1. 页面设计

当跳转到注册界面时，用户需要在注册界面输入一些相关信息，针对这种情况，我们可以使用 TableLayout 方便地组织页面，所使用的组件如表 9-5 所示。

表 9-5　注册界面组件表

类　型	ID	含　义	类　型	ID	含　义
TextView	Name	提示用户输入用户名	TextView	Sex	提示用户输入性别
EditText	Name_in	用户名输入框	EditText	Sex_in	性别输入框
TextView	Pass	提示用户输入密码	TextView	Hobby	提示用户输入爱好
EditText	Pass_in	密码输入框	EditText	Hobby_in	爱好输入框
TextView	Age	提示用户输入年龄	Button	okBtn	“确定”按钮
EditText	Age_in	年龄输入框			

2. 整体设计

在整体设计部分的主要工作是得到组件的操作对象，并设置“确定”按钮的监听事件：

```
package com.wes.Chapter9;

import ...                                   //省略部分导入
import android.content.ContentValues;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

public class RegisterActivity extends Activity
{
    String name ;
    String pass ;
    String sex ;
    String age ;
    String hobby ;
    SQLiteDatabase db;
     @Override
       public void onCreate(Bundle savedInstanceState)
       {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.register);
final EditText et1 = (EditText) findViewById(R.id.name_in);
            final EditText et2 = (EditText) findViewById(R.id.pass_in);
            final EditText et3 = (EditText) findViewById(R.id.sex_in);
            final EditText et4 = (EditText) findViewById(R.id.age_in);
            final EditText et5 = (EditText) findViewById(R.id.hobby_in);
            Button okBtn  = (Button) findViewById(R.id.okBtn);
            okBtn.setOnClickListener(new OnClickListener()
               {
                    //完成“确定”按钮功能
               });
       }
}
```

3．“确定”按钮功能设计

完成“确定”按钮功能时需要如下 4 个步骤：

（1）得到用户输入的信息。

（2）得到数据库对象。

（3）判断用户名是否已存在，不存在则将信息保存到数据库中，否则提示已存在该用户。

（4）保存成功后，跳转到登录成功页面。

```
@Override
                   public void onClick(View arg0)
                   {
                       //得到用户输入的信息
                       name = et1.getText().toString();
                       pass = et2.getText().toString();
                       sex  = et3.getText().toString();
                       age  = et4.getText().toString();
                       hobby= et5.getText().toString();
                       //得到数据库对象
                       DatabaseHelper helper = new DatabaseHelper
                        (getBaseContext());
                       db = helper.getWritableDatabase();
```

```
                        //判断该用户名是否已经存在
Cursor cursor = db.query(DatabaseHelper.TABLENAME,
                                new String[]{DatabaseHelper.NAME},
                                DatabaseHelper.NAME + "=?",
                                new String[]{name}, null, null, null);
                        //若查询到记录则提示"已存在"
if (cursor.getCount()>0)
                        {
                            Toast.makeText(getBaseContext(), "该用户已存
                            在! ! ", Toast.LENGTH_LONG).show();
                            return;
                        }
                        //若不存在则插入数据
                        ContentValues values = new ContentValues();
                        values.put(DatabaseHelper.NAME, name);
                        values.put(DatabaseHelper.PASSWORD, pass);
                        values.put(DatabaseHelper.SEX, sex);
                        values.put(DatabaseHelper.AGE, age);
                        values.put(DatabaseHelper.HOBBY, hobby);
                        db.insert(DatabaseHelper.TABLENAME, null, values);
                        //进行跳转
                        Intent i = new Intent(RegisterActivity.this,
                        SucActivity.class);
                        i.putExtra("name", name);
                        startActivity(i);
                    }
```

注意：*在跳转时需要将参数用户名传递给登录成功界面，否则登录成功界面无法进行查询。与登录界面不同的是，这里得到的是可写的数据库对象，而之前得到的仅仅是可读的数据库对象。*

9.4.5　登录成功界面设计

登录成功界面的主要工作有两类：首先需要根据传递进来的用户名对数据库进行查询，接着将查询的结果相应地显示到界面中。

1．页面设计

界面设计时显示用户的具体信息，需要 5 个 TextView 组件进行相应的显示。这里就不再一一罗列，读者可以自由发挥，进行页面设计部分。

2．整体设计

在 onCreate()方法中调用了 3 个方法分别进行界面初始化、数据库查询以及结果显示：

```
package com.wes.Chapter9;

import ...                              //省略部分导入
import android.content.ContentValues;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
```

```
public class SucActivity extends Activity
{
    String name ;
    String pass ;
    ...                                //省略部分对象声明
    TextView name_in;
    TextView pass_in;
    ...                                //省略部分组件声明
    SQLiteDatabase db;
     @Override
       public void onCreate(Bundle savedInstanceState)
       {
          super.onCreate(savedInstanceState);
           setContentView(R.layout.welcome);
           initView();                 //初始化界面
           doQuery();                  //进行查询
           doShow();                   //进行显示
       }
```

3. 初始化页面

这一部分相当简单了，这里就不再赘述。

```
public void initView()
{
     btn = (Button) findViewById(R.id.back);
     name_in = (TextView) findViewById(R.id.name_in);
     pass_in = (TextView) findViewById(R.id.pass_in);
     sex_in = (TextView) findViewById(R.id.sex_in);
     age_in = (TextView) findViewById(R.id.age_in);
     hobby_in = (TextView) findViewById(R.id.hobby_in);

     btn.setOnClickListener(new OnClickListener()
       {
            @Override
            public void onClick(View v)
            {
                Intent i = new Intent(SucActivity.this,LoginActivity.
                class);
                startActivity(i);
            }
       });
}
```

4. 进行查询

根据得到的用户名查询该用户的密码、年龄、性别和爱好等信息：

```
public void doQuery()
{
     name = getIntent().getExtras().getString("name");
        //得到数据库对象
        DatabaseHelper helper = new DatabaseHelper(getBaseContext());
        db = helper.getReadableDatabase();
        //准备查询的属性
        String[] columns = new String[]{
                DatabaseHelper.PASSWORD,
                DatabaseHelper.AGE,
```

```
                DatabaseHelper.SEX,
                DatabaseHelper.HOBBY};

        Cursor cursor = db.query(DatabaseHelper.TABLENAME, columns,
        DatabaseHelper.NAME + "=?", new String[]{name}, null, null, null);
        cursor.moveToFirst();           //指向第一条记录
        while(!cursor.isAfterLast())    //判断是否是最后一条记录
        {
            pass = cursor.getString(0);
            age = cursor.getString(1);
            sex = cursor.getString(2);
            hobby = cursor.getString(3);
            cursor.moveToNext();        //指向下一条记录
        }
}
```

5．进行显示

这一部分对于读者朋友应该非常简单了，相关代码这里就省略了，读者如有疑问可以参考随书的源代码，最后运行效果显示如图 9.9 所示。



登录界面，提示用户名不存在



登录成功界面



注册界面



提示用户已存在

图 9.9　运行效果图

9.5　使用 ContentProvider 共享数据

众所周知，Android 中所有的数据都是私有的，那么难道我们就不可以共享数据么？显然，如此流行的 Android 不会让我们失望，ContentProvider 横空出世。顾名思义，内容提供者就是将私有的数据暴露给其他的使用者，如通话记录、联系人列表等。当然，我们还可以自己定义 ContentProvider，将自定义的数据提供给其他的应用程序使用。

9.5.1　了解 ContentProvider

1. 什么是ContentProvider

ContentProvider 机制可以帮助开发者在多个应用中操作数据，包括存储、修改和删除等。这也是在应用程序间共享数据的唯一方式。一个 ContentProvider 类实现了一组标准的接口，它们是：

（1）ContentProvider.insert(Uri arg0, ContentValues arg1)

（2）ContentProvider.query(Uri arg0, String[] arg1, String arg2, String[] arg3, String arg4)

（3）ContentProvider.update(Uri arg0, ContentValues arg1, String arg2, String[] arg3)

（4）ContentProvider.delete(Uri arg0, String arg1, String[] arg2)

（5）ContentProvider.getType(Uri arg0)

通过这些接口，其他的应用程序可以很方便地对其数据进行操作，而无需关心数据结构，无论是数据库还是文本文件或者是音频文件等等。

通过本段讲解，读者再来细细品味 ContentProvider 这个名字，不难发现其名称还是取得非常贴切的。它的作用不就是把本身的内容提供给其他的程序来使用么？所以它被称作“内容提供者”还真是名副其实了。

2. 什么是URI

细心的读者不难发现，在之前我们提到的 5 个接口中都出现了“Uri”一词，那么 Uri 这个类到底是什么作用呢？我们又该如何使用呢？

URI 是 Universal Resource Identifier 的缩写，也就是通用资源标志符的意思。它的作用就是告诉使用者，数据的具体位置，所以在 URI 中一定包含数据的路径。事实上，在 Android 的 Uri 中主要包括 3 个部分：

（1）“content:// ”，Android 命名机制规定所有的内容提供者 Uri 必须以“content://”开头。

（2）数据路径，正如前文所说，通过该路径，其他的应用程序可以顺利地找到具体数据。

（3）ID，这个是可选的，如果不填表示取得所有的数据。

举几个例子也许可以更形象地说明问题：

（1）content://contacts/peopl：这个 Uri 的意思是所有的联系人信息，因为它最后没有包

含具体的 ID。

（2）content://contacts/people/1：这个 Uri 的意思是 ID 为 5 的联系人的信息。

（3）content://media/externa：这个 Uri 的意思是所有的媒体信息。

当然这些长长的 Uri 在很多时候我们并不需要接触它，因为“贴心”的 Android 已经帮助我们定义了一些常量用来代替这些 Uri。

3. ContentResolver

既然我们知道 ContentProvider 可以将私有数据暴露给其他的应用程序，那么我们怎么获得这些数据呢？这个时候我们就需要使用 ContentResolver 了。Android 的数据共享机制中，ContentProvider 是作为提供者出现，而 ContentResolver 则作为消费者出现。通过 getContentResolver()可以得到当前应用的 ContentResolver 对象。

要实现一个 ContentResolver 同样需要实现 5 个接口，与 ContentProvider 一一对应：

（1）ContentResolver. delete(Uri url，String where，String[] selectionArgs)

（2）ContentResolver .update(Uri uri，ContentValuesvalues，Stringwhere，String[]selectionArgs)

（3）ContentResolver.query(Uriuri，String[]projection，Stringselection，String[]selectionArgs，String sortOrder)

（4）ContentResolver. insert(Uri url，ContentValues values)

（5）ContentResolver. getType(Uri url)

可能现在大家看这些接口觉得有些眼熟，没错，这些接口和我们之前讲解的数据库操作非常类似，甚至连查询的返回值都一样，两者都是以光标 Cursor 的形式返回。所以读者在理解本小节时要与之前学习的数据库相关知识结合起来，这样可以事半功倍。

9.5.2　使用 ContentProvider

Android 系统本身就包含了一些内建的程序，包括联系簿以及通话记录等。这些应用程序往往都作为 ContentProvider 向外界提供数据，我们可以方便地使用 managedQuery()方法查询相关数据。为了更好地说明如何使用 ContentProvider，我们通过以下的几个实例来实践一下。

1. 联系簿

首先我们要向手机中添加一些联系人方式，打开虚拟机的电话簿，单击 menu 按钮后显示如图 9.10 所示。

单击 New contact 按钮，在弹出的如图 9.11 的对话框中输入相关信息。完成后单击 Done 按钮，这样就已经成功地创建了联系人，你可以多创建几个，这样我们的程序运行起来效果可能更好一些。

接下来，我们新建一个 Android 工程，将继承的 Activity 改为 ListActivity，这样我们的代码可以更简洁，当然你可以选择自定义的界面，也支持大家这样做。因为这样在学习新知识的同时还可以回顾第 7 章关于 ListView 的知识。这里为了代码的简练，着重突出 ContentProvider 的使用就选择了继承 ListActivity。



图 9.10　联系簿



图 9.11　填写信息

接下来我们只需要 3 个步骤：

（1）查询联系人列表获得 Cursor 对象，方法为：

```
managedQuery(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder)
```

这里有 5 个参数，其分别为：

- Uri：指定数据的具体路径。
- projection：要查询的属性列。
- selection：条件，相当于 WHERE 子句，如“id=？”。
- selectionArgs：条件的参数，用来代替条件中的“？”。
- sortOrder：排序，升序为 ASC，降序为 DESC。

（2）新建 Adapter：

```
ListAdapter adapter = new SimpleCursorAdapter(Context context, int layout,
Cursor c, String[] from, int[] to)
```

关于 Adapter 的创建在前文已经有过讲解，这里就不再叙述，如果仍有疑问可以翻看第 7 章。

（3）设置 Adapter：

```
setListAdapter(adapter);
```

通过该方法将 Adapter 与 ListView 绑定起来。

按照以上步骤，本实例的代码片段为：

```
@Override
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
 Cursor c1 = this.managedQuery(ContactsContract.CommonDataKinds.Phone.
 CONTENT_URI,null,null, null, null);
        ListAdapter adapter3 = new SimpleCursorAdapter(getApplication-
        Context(),
                  android.R.layout.simple_list_item_2, c1,
newString[]{ContactsContract.Contacts.DISPLAY_NAME,
ContactsContract.CommonDataKinds.Phone.NUMBER},
                  new int[]{android.R.id.text1,android.R.id.text2});
        setListAdapter(adapter3);
}
```

这里需要说明的是：前文提过，由于 Android 系统中 Uri 往往很长，这给使用带来了不便，所以 SDK 为开发者定义了一些常量用来代替 Uri，这里 ContactsContract.Common-DataKinds.Phone.CONTENT_URI 就是具体的 Uri 了。在 Android 1.X 版本时，联系簿的 Uri 被包含在 Contacts.People 中，不过在 2.0 以后的版本中，SDK 对其进行了升级，开发了一个新的类——ContactsContract，所有与 Contacts 相关的信息在其中都可以找到。而以前的 Contacts 类已经不被建议使用了。

在 Adapter 中的两个属性值，顾名思义，一个是显示的名字，一个是号码。当然不要忘记了添加权限：

```
<uses-permission android:name="android.
permission.READ_CONTACTS" />
```

赶紧运行一下，效果如图 9.12 所示。

如果你有兴趣，你可以继续扩展该应用，如单击其中的选项就拨打电话等。

2．通话记录

同样地，利用以上步骤我们可以查询通话记录，步骤是一样的，只是其中的 Uri 与属性不同，其代码片段为：

```
@Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       Cursor c = this.managedQuery(CallLog.Calls.CONTENT_URI,null,null,
       null, null);
        ListAdapter adapter = new SimpleCursorAdapter(getApplication-
        Context(),
        android.R.layout.simple_list_item_2, c,
        new String[]{Calls.NUMBER,Calls.DURATION},
        new int[]{android.R.id.text1,android.R.id.text2});

       setListAdapter(adapter);
   }
```

注意这里的加粗部分，这是本实例与之前的联系簿不同的地方，首先它的 Uri 是 CallLog.Calls.CONTENT_URI，属性为号码和持续时间。当然还有更多其他的信息，如日期、id、次数等等，运行效果如图 9.13 所示。

3．多媒体信息

如果要查询多媒体信息只需将 Uri 改成 MediaStore 的子类下的常量就可以了，如：

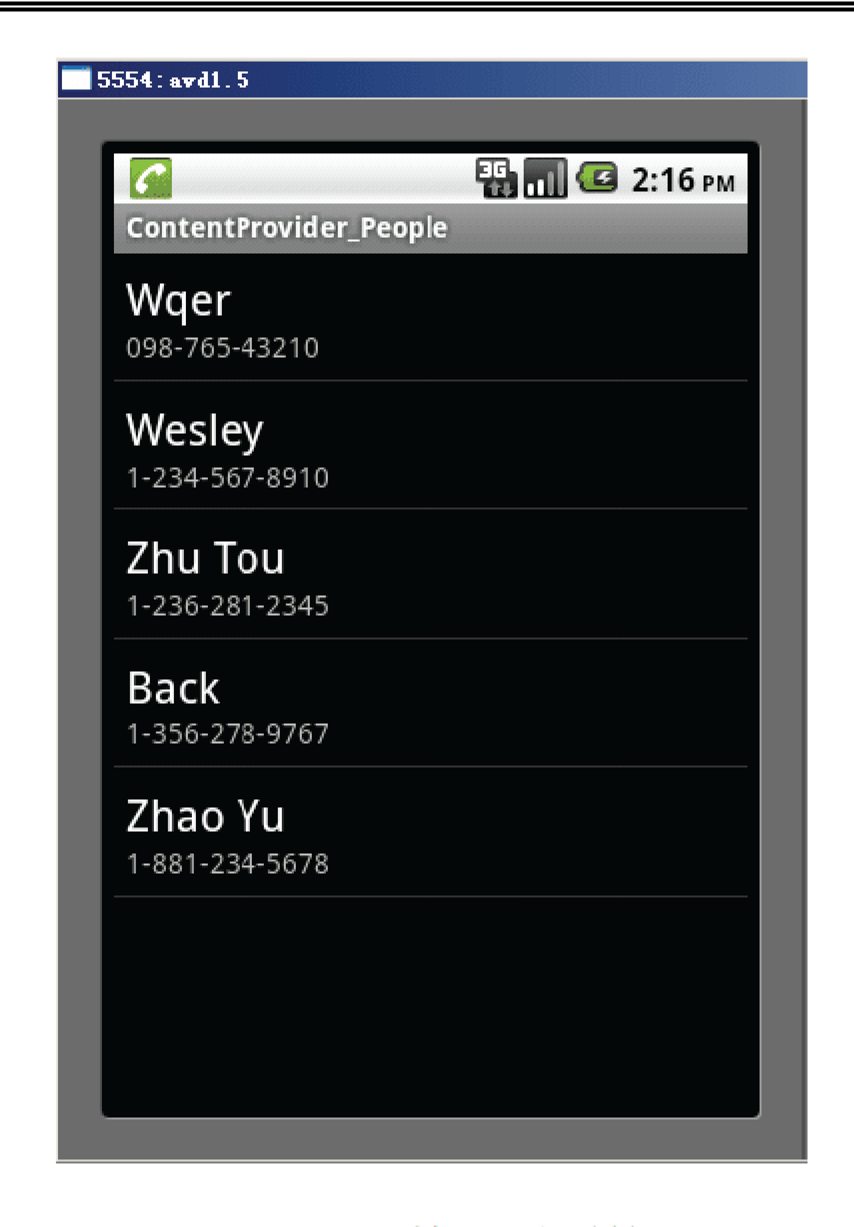


图 9.12 简易联系簿



图 9.13 通话记录

- MediaStore.Audio.Media.EXTERNAL_CONTENT_URI：音频文件的 URI
- MediaStore.Video.Media.EXTERNAL_CONTENT_URI：视频文件的 URI
- MediaStore.Images.Media.EXTERNAL_CONTENT_URI：图片文件的 URI

我们以音频文件为例，查询文件名和持续时间：

```
@Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      Cursor c = this.managedQuery(MediaStore.Audio.Media.EXTERNAL_
      CONTENT_URI,null,null, null, null);
      String tittle = MediaStore.Audio.Media.TITLE;
      String duration = MediaStore.Audio.Media.DURATION;
      ListAdapter adapter = new SimpleCursorAdapter(getApplica-
      tionContext(),
       android.R.layout.simple_list_item_2, c,
       new String[]{tittle,duration},
       new int[]{android.R.id.text1,android.R.id.text2});

      setListAdapter(adapter);
}
```

在运行前你需要向模拟机中添加一些音频文件，否则查询不到信息会造成异常。运行显示如图 9.14 所示。

4．浏览器书签

Android 系统提供了一组用于查询浏览的历史记录和书签的 Uri，我们可以将之用户数据统计等，要使用它，我们只需将之前实例的 Uri 改为：

```
Browser.BOOKMARKS_URI
```

我们同样可以查询浏览的历史站点的名称和次数。其属性分别为：

```
Browser.BookmarkColumns.TITLE  标题
Browser.BookmarkColumns.VISITS 浏览次数
```

当然还有更多的属性，如创建时间 Browser.BookmarkColumns.VISITS 等。这样，我们的代码就变成了：

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Cursor c = this.managedQuery(Browser.BOOKMARKS_URI,null,null, null,
        null);
        String tittle = Browser.BookmarkColumns.TITLE;
        String vis = Browser.BookmarkColumns.VISITS;
        ListAdapter adapter = new SimpleCursorAdapter(getApplica-
        tionContext(),
         android.R.layout.simple_list_item_2, c,
         new String[]{tittle,vis},
         new int[]{android.R.id.text1,android.R.id.text2});

        setListAdapter(adapter);
```

最后，不要急着运行，忘记了添加如下的权限，我们的程序可运行不了哦：

```
<uses-permission android:name="com.android.browser.permission.READ_
HISTORY_BOOKMARKS"/>
```

现在运行一下试试看，效果如图 9.15 所示。



图 9.14　音频列表



图 9.15　历史记录

到这里，常用的一些系统 Uri 就介绍得差不多了。当然，限于篇幅，这里只是做了一些最简单的介绍，如果需要深入开发还需读者朋友们再仔细研究。下一小节我们将讲解如何使用 ContentResolver 操作 ContentProvider 中的数据。

9.5.3　使用 ContentResolver

前文我们已经说明，ContentResolver 是作为一个消费者出现的，那么我们应该怎样进行消费呢？换句话说，ContentResolver 怎样使用呢？本小节仅以 Contacts 为例对 ContentResolver 的使用进行讲解，关于其他种类的 ContentProvider 则交由读者自行学习，

因为它们之间大同小异。接下来，让我们从最简单的开始。

1. 删除数据

首先，我们要得到 ContentResolver 的对象：

```
ContentResolver resolver =getContentResolver();
```

接着，使用 delete()方法可以很方便地删除数据，其语法格式如下：

```
ContentResolver.delete(Uri url, String where, String[] selectionArgs)
```

这里有 3 个参数，其分别为

- Uri：你需要操作的 ContentProvider 的 Uri，如联系簿的数据的 Uri 是 Data.CONTENT_URI。
- where：条件，若为 null 则表示全部删除。
- selectionArgs：条件参数，用来替代条件中的“？”。

如要删除所有联系人，语法为：

```
resolver.delete(Data.CONTENT_URI, null, null);
```

如果在真机的环境下运行，要执行这句话请一定要慎重，因为一不小心你就会和你的朋友们失去联系了。如果要删除某条记录，则在条件中加入参数，如要删除名字为“WES”的记录：

```
resolver.delete(Data.CONTENT_URI, StructuredName.DISPLAY_NAME+"=?", new
String[]{"WES"});
```

2. 查询数据

ContentResolver 的查询数据与 managedQuery 方法类似：

```
ContentResolver.query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder)
```

同样的，这里有 5 个参数，其分别为

- Uri：要访问的 ContentProvider 的地址。
- projection：要查询的属性，为 null 则查询所有信息。
- selection：条件，为 null 表示所有记录。
- selectionArgs：选择参数，用来代替第三个参数的“？”。
- sortOrder：排序，升序为 ASC，降序为 DESC。

如查询所有联系人的信息：

```
resolver.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,null,
null, null, null);
```

3. 更新数据

更新数据的方法同样非常简单：

```
ContentResolver.update(Uri uri, ContentValues values, String where, String[]
selectionArgs)
```

这里有 4 个参数，其分别为

- ❑ uri：要访问的 ContentProvider 的地址。
- ❑ values：ContentValues 对象，数据的映射，在前文已经讲解过，用来存储键值对。
- ❑ where：条件子句，需要使用的参数用“？ ”代替。
- ❑ selectionArgs：条件参数，用来代替 where 子句中的“？ ”。

例如，要将名称为“WES”的联系人的姓名改为“wes”，其语法格式为：

```
values.put(StructuredName.DISPLAY_NAME, "wes");
resolver.update(Data.CONTENT_URI,                                values,
StructuredName.DISPLAY_NAME+"=?", new String[]{"WES"});
```

4．插入数据

插入的方法看上去同样非常简单：

```
ContentResolver.insert(Uri url, ContentValues values)
```

这里只有两个参数：一个是 Uri，一个是 ContentValues。也许读者觉得很简单啊，为什么要放在最后讲解呢？事实上，插入的方法不是难题，难点是你需要插入的数据的数据结构。我们仅以 Contacts 为例：要求很简单，插入一条 Name 为“wes”，Number 为“123456789”的记录。

这个时候也许很多读者依然在想，还是没有什么困难的啊，也许你会想出如下的代码：

```
ContentValues values = new ContentValues();
Values.put("Phone.NAME","wes");
Values.put("Phone.NUMBER","123456789")
Resolver.insert(Phone.CONTENT_URI,values);
```

很可惜的是，这样的代码运行之后被证明是错误的，读者朋友们如果不信可以尝试一下。事实上，Contacts 有其自己的数据结构。

首先，我们要得到要插入数据的 Uri，方法为插入一条空的 ContentValues：

```
ContentValues values = new ContentValues();
Uri rawContentUri = resolver.insert(RawContacts.CONTENT_URI, values);
```

RawContacts 表存储了联系人的 Id。通过上述方法我们可以得到返回值 RawContacts 的 Uri。接着，通过该 Uri 我们可以解析出具体的记录 Id：

```
long rawContentId = ContentUris.parseId(rawContentUri);
```

然后，得到了具体的 ID，我们就可以像 Data 表插入实际的数据了，先插入姓名：

```
values.clear();
      values.put(Data.RAW_CONTACT_ID, rawContentId);
      values.put(Data.MIMETYPE, StructuredName.CONTENT_ITEM_TYPE);
      values.put(StructuredName.DISPLAY_NAME, "WESLEY");
      resolver.insert(ContactsContract.Data.CONTENT_URI, values);
```

Data 表中存储了所有的联系人数据，其中就包括最重要的姓名和号码，同理，插入号码的方法如下：

```
values.clear();
      values.put(Data.RAW_CONTACT_ID, rawContentId);
```

```
        values.put(Data.MIMETYPE, Phone.CONTENT_ITEM_TYPE);
        values.put(Phone.NUMBER, "12345678901");
        values.put(Phone.TYPE, Phone.TYPE_MAIN);
        resolver.insert(Data.CONTENT_URI, values);
```

所以，综上所述，插入数据并没有读者朋友们想象的那么简单，重要的是先理解其数据结构，只有真正洞悉了其中的结构，插入数据时才能做到有的放矢。

系统自带的 ContentProvider 我们就讲解到这里，下一节我们将学习编写自定义的 ContentProvider，将自己的数据提供给其他的应用程序。

9.6　自定义 ContentProvider

仅仅是使用系统自带的 ContentProvider 是不是让你觉得还不是很“给力”？如果我们的应用中同样有需要被共享的数据，那么能不能自己成为 ContentProvider 呢？要成为一个 ContentProvider 需要实现一组标准的接口，完成后还需在 AndroidManifest 文件中进行注册。

9.6.1　ContentProvider 需要实现的接口

既然要实现 ContentProvider，那么一定要拥有自己的数据，为了简便，这里就利用 9.4 节中实例的数据来完成 ContentProvider 的讲解。

首先，我们从直观上了解 ContentProvider 的结构，以便梳理我们即将要进行的工作，在 DatabaseDemo_2 工程中新建一个 Class 文件，命名为 MyProvider，继承自 ContentProvider，那么它的结构是：

```
public class MyProvider extends ContentProvider
{
    @Override
    public int delete(Uri arg0, String arg1, String[] arg2)
    {
        return 0;
    }

    @Override
    public String getType(Uri arg0)
    {
        return null;
    }

    @Override
    public Uri insert(Uri arg0, ContentValues arg1)
    {
        return null;
    }

    @Override
    public boolean onCreate()
    {
        return false;
    }

    @Override
```

```
    public Cursor query(Uri arg0, String[] arg1, String arg2, String[] arg3,
    String arg4)
    {
        return null;
    }

    @Override
    public int update(Uri arg0, ContentValues arg1, String arg2, String[]
    arg3)
    {
        return 0;
    }

}
```

换言之，只要我们实现了以上的 6 个方法，我们就完成了数据从私有到共享的转变。

9.6.2　实现 ContentProvider

在实现以上的 6 个方法前，我们还需要做一件事——定义常量。或者准确地说，这是两件事：第一，定义 Uri，第二，定义数据列名。

1．定义Uri

众所周知，要访问 ContentProvider 必须使用 Uri 来寻找到其具体的位置，那么作为一个 ContentProvider，我们必须提供一个 Uri，名称为 CONTENT_URI。且以“content://”开头。一般情况下，它包含 3 个部分：

- ❑ 头部：content://。
- ❑ 授权：authority，一般可以填写完整的类名，以保证唯一。
- ❑ 表名：你需要暴露的数据的表名，该部分可以不填写。

这里，我们将其 Uri 定义为：

```
public static final String AUTHORITY       = "com.wes.Chapter8.
MyProvider";                                              //授权
public static final Uri CONTENT URI = Uri.parse("content://" + AUTHORITY+
"/"+DatabaseHelper.TABLENAME);
```

2．定义数据列名

所有使用该 Provider 的用户都必须知道其提供的数据究竟有哪些。所以我们必须要在类中加以定义，本实例中我们可以提供如下的数据列以供查询：

```
public final static String NAME         = "name";
public final static String SEX          = "sex";
public final static String AGE          = "age";
public final static String HOBBY        = "hobby";
public final static String PASSWORD     = "password";
```

3．实现query()方法

一般情况下，query()方法是其他程序对 ContentProvider 暴露的数据使用最多的接口了，我们就从 query()方法开始逐步实现其 5 个接口。

ContentProvider 的接口标准规定，query()方法的返回值必须是一个 Cursor 对象，事实上，该 query()方法的实现只是对 SQLiteQueryBuilder.query()方法的再封装。

接下来我们就开始具体的实现步骤：由上一小节我们知道，query()方法中包含有参数 Uri，首先我们需要知道其 Uri 具体的指向，是执行批量操作还是执行指定 Id 的操作，而要得到该信息，我们必须学习使用一个新的类：Uri 匹配器——UriMatcher，实现步骤为：

```
private static final int    USERINFOS       = 1;
    private static final int    USERINFO_ID     = 2;
    private static final UriMatcher MATCHER  = new UriMatcher(UriMatcher.
    NO_MATCH);
    static
    {
        MATCHER.addURI(AUTHORITY, DatabaseHelper.TABLENAME, USERINFOS);
        MATCHER.addURI(AUTHORITY, DatabaseHelper.TABLENAME+"/#",
        USERINFO_ID);
    }
```

我们可以分 3 步看：

（1）我们要定义两个常量用以区分 Uri 的类型，这里的 USERINFOS 和 USERINFO_ID 就是两种状态。

（2）得到 UriMatch 对象，对于所有类型我们在参数中填写 UriMatcher.NO_MATCH。

（3）为 UriMatcher 添加 Uri 匹配信息，第一个参数是授权，第二个参数是路径，第三个参数则是具体的状态标志了。我们看到这里使用了#，其作用是代替一个数字，同样地，我们还可以使用*表示任何文字。

得到了 UriMatcher 对象以后，我们就可以执行具体的查询工作了，前文曾经提到，query()方法往往是 SQLiteQueryBuilder.query()方法的再封装。那么接下来我们要做的工作就比较清楚了：

（1）得到 SQLiteQueryBuilder 对象。

（2）为 SQLiteQueryBuilder 对象设置表。

（3）通过 UriMatcher 对象对 Uri 进行匹配，如果是针对具体的 Id，则在条件中添加 Id 相关条件；如果不是，则直接将传递给 ContentProvider.query()方法的参数填写到 SQLiteQueryBuilder.query()方法中。

（4）将查询的结果作为返回值返回。

按照以上步骤，具体代码如下：

```
    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder)
    {
        SQLiteQueryBuilder builder = new SQLiteQueryBuilder();
        builder.setTables(DatabaseHelper.TABLENAME);
        int match = MATCHER.match(uri);
        if (match == USERINFO_ID)
        {
            builder.appendWhere("_id="+ uri.getLastPathSegment());
        }
        Cursor c = builder.query(db, projection, selection, selectionArgs,
        null, null, sortOrder);

        c.setNotificationUri(resolver, uri);   //保持数据同步
```

```
        return c;
    }
```

我们还需要注意加粗部分，其作用是保持数据的同步性，因为同时可能有多个线程在操作表，在查询结束后可能会发生源数据改变的情况，这个时候我们就需要调用：

```
Cursor.setNotificationUri(ContentResolver arg0, Uri arg1)
```

来观察哪个 Uri 的数据修改，以此达到数据同步的目的。

4．实现insert()方法

Insert()方法的作用是插入记录，其参数有两个：一个是 Uri，另一个是要插入的数据 ContentValues。插入方法的实现比较简单，只需简单调用方法：

```
SQLiteDatabase.insert(String table, String nullColumnHack, ContentValues
values)
```

这里的第二个参数可以设为空。执行结果可以作为返回值返回。

但是，请各位注意，事情并没有这么简单，我们还有两件重要的事情要做：

（1）判断 Uri 是否合法。我们要保证，这里就是需要插入数据的正确地方。例如，一个指向具体记录的 Uri 就是不合法的，因为指向特定记录后根本无法插入。

（2）通知数据改变。通过：

```
ContentResolver.notifyChange(Uri uri, ContentObserver observer)
```

可以达到目的。

按照以上的步骤，insert()方法的具体代码为：

```
@Override
public Uri insert(Uri uri, ContentValues values)
{
    int match = MATCHER.match(uri);
    if (match != USERINFOS)
    {
        throw new IllegalArgumentException("URI 错误! ");
    }
    long id = db.insert(DatabaseHelper.TABLENAME, null, values);
    if (id > 0)
    {
        Uri newUri = ContentUris.withAppendedId(uri, id);
        resolver.notifyChange(newUri, null);
        return newUri;
    }
    return null;
}
```

这里我们又使用了一个新的类：ContentUris。本段代码中我们使用了 ContentUris.withAppendedId(uri, id)方法，其作用是将新插入的 ID，添加到 Uri 中。添加完毕后，再通过 ContentResolver.notifyChange(Uri uri, ContentObserver observer)方法通知系统底层数据发生了改变。ContentUris 类还有一个使用的方法就是 ContentUris.parseId(Uri uri)，通过该方法可以得到传递进来的 uri 指向的具体 ID。

5. 实现update()方法

更新数据的方法我们同样分为以下步骤：

（1）Uri 匹配，确定 Uri 指向表还是表中的特定记录。

（2）如果是执行特定记录，在条件中添加 id=? 。如果不是则直接使用参数执行更新。

（3）通知数据改变。

按照以上步骤，其功能代码为：

```
@Override
public int update(Uri uri, ContentValues values, String selection, String[]
selectionArgs)
{
    int match = MATCHER.match(uri);
    int updated;

    switch (match)
    {
    case USERINFO_ID:
    {
        String id = uri.getLastPathSegment();
        if (TextUtils.isEmpty(selection))
        {
            updated = db.update(DatabaseHelper.TABLENAME, values, " id="+
            id, null);
        }
        else
        {
            updated = db.update(DatabaseHelper.TABLENAME, values,
            selection + "and" + _ID + "=" + id, selectionArgs);
        }
        break;
    }
    case USERINFOS:
    {
        updated = db.update(DatabaseHelper.TABLENAME, values, selection,
        selectionArgs);
        break;
    }
    default:
        throw new IllegalArgumentException("URI 错误! ");
    }
    resolver.notifyChange(uri, null);
    return updated;
}
```

注意在判断出 Uri 指向某条记录后，我们还需再执行一个判断，判断条件是否为“”，如果是“”，则直接将条件设为 id=? ，如果不为“”，则通过 and 关键词将 id=? 条件附加上。

6. 实现delete()方法

与 update()方法类似，delete()方法只需在 update()方法的结构基础上稍作修改。修改后其代码如下：

```
@Override
public int delete(Uri uri, String selection, String[] selectionArgs)
{
    int match = MATCHER.match(uri);
    int updated;

    switch (match)
    {
    case USERINFO ID:
    {
        String id = uri.getLastPathSegment();
        if (TextUtils.isEmpty(selection))
        {
            updated = db.delete(DatabaseHelper.TABLENAME, "_id="+id, null);
        }
        else
        {
            updated = db.delete(DatabaseHelper.TABLENAME,selection + "and"
            + _ID + "=" + id, selectionArgs);
        }
        break;
}
case USERINFOS:
{
    updated = db.delete(DatabaseHelper.TABLENAME,  selection,
    selectionArgs);
        break;
    }
    default:
        throw new IllegalArgumentException("URI 错误! ");
    }
    resolver.notifyChange(uri, null);
    return updated;
}
```

7. 实现getType()方法

该方法的作用是对传递进来的 Uri 参数进行判断，最后将类型返回。按照 Android SDK 的指导，我们使用了 CURSOR_ITEM_BASE_TYPE 和 CURSOR_DIR_BASE_TYPE 来区分针对特定 Id 的类型和针对目录的类型。其代码如下：

```
Public final static String CONTENT ITEM TYPE = ContentResolver.
CURSOR ITEM BASE TYPE +"/"+DatabaseHelper.TABLENAME;
public final static String CONTENT TYPE = ContentResolver.CURSOR
DIR BASE TYPE+"/"+DatabaseHelper.TABLENAME;

    @Override
    public String getType(Uri uri)
    {
        int match = MATCHER.match(uri);
        if (match == USERINFO ID)
        {
            return CONTENT_ITEM_TYPE;
        }
        else if (match == USERINFOS)
        {
            return CONTENT_TYPE;
        } else
        {
```

```
            throw new IllegalArgumentException("URI 错误！");
        }
    }
```

9.6.3　更新 AndroidManifest 文件

实现完 ContentProvider 文件后不要忘记更新注册文件，否则你辛辛苦苦编写出来的代码将无法发挥作用，provider 的注册需要两个部分：（1）类名，（2）授权。其语法为：

```
<provider android:name="com.wes.Chapter8.MyProvider"
    android:authorities="com.wes.Chapter8.MyProvider"
    android:multiprocess="true"/>
```

这里比之前讲解的两个属性还要多出一个 multiprocess，其意义为多线程，作用是允许两个或两个以上的线程同时访问该 ContentProvider 中的内容，这也很好地说明了之前我们做的一些数据同步工作的重要性。

最后，还需着重强调其在注册文件中添加的位置，与允许权限不同的是：

```
<uses-permission android:name="android.permission.WRITE_APN_SETTINGS" />
```

类似这样的语句是必须填写在<application>< /application >节点外的，而 provider 的注册位置则必须在<application>< /application >节点内。例如，最后本实例的注册文件如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.wes.Chapter8"
     android:versionCode="1"
     android:versionName="1.0">
   <uses-sdk android:minSdkVersion="7" />

   <application android:icon="@drawable/icon" android:label=
   "@string/app_name">
      <activity android:name="LoginActivity"
              android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
    <activity   android:name="RegisterActivity"
              android:label="@string/reg name">
      </activity>
      <activity    android:name="SucActivity"
              android:label="@string/suc name">
      </activity>
    <provider android:name="com.wes.Chapter8.MyProvider"
    android:authorities="com.wes.Chapter8.MyProvider"
    android:multiprocess="true"/>
   </application>
</manifest>
```

注意：其中的代码部分，如果添加位置错误的话，系统将无法识别 ContentProvider，如果你使用 ContentResolver 执行查询，系统将无法根据你的 Uri 找到具体的数据，这一点很重要。

9.7　小　　结

本章讲解了 Android 编程过程中的一些数据操作，包括 SharedPreferences、文件存储以及 SQLite 数据库存储。还讲解了将私有数据共享给其他应用的方法——ContentProvider。本章重点是轻量级数据库的学习，包括查询、插入、更新、删除等操作。难点是自定义 ContentProvider 的编写，希望读者可以反复阅读本章加强理解。下一章，我们将讲解多媒体的相关知识，包括拍照、音频以及视频的使用方法。

第 10 章　绚丽的多媒体技术

多媒体——无论是声音、图像抑或是视频——可以使你的应用更加绚丽多彩。如今的手机设备至少含有一个麦克风——我们可以通过它录制个性铃音，至少含有一个摄像头——我们可以通过它拍摄图片或者录制视频。Android 提供了一系列的多媒体 API 供开发者使用，甚至包括了文件输出格式和编码格式的控制。在本章中，读者将学会使用麦克风和摄像头开发出精彩的多媒体应用。

10.1　简单处理音频

音频处理是 Android 开发的一个大课题。在本节中将带领大家学习一些在开发中常用的 API，以及必须要了解的一些基础知识。但本书篇幅有限，更多的知识需要读者自己去深入了解，只有知其然更知其所以然才能开发出更优秀、更高效的应用。

10.1.1　使用 MediaRecoder 录制音频

在大多数的开发中，开发者使用 MediaRecorder 录制声音文件。因为它使用非常方便、简单而又安全，可以满足大部分开发的需求。要使用 MediaRecorder 对象需要如下几个步骤：

（1）实例化一个 MediaRecorder 对象。

（2）设置采集设备。

（3）设置录制格式。

（4）设置文件输出格式。

（5）设置目标文件。

（6）准备录制。

（7）开始录制。

（8）停止录制。

（9）释放资源。

接下来将为大家详细讲解每个步骤的功能。

1．实例化一个MediaRecorder对象

通过 new MediaRecorder()方法获得一个 MediaRecorder 对象，以便对其进行设置和操作。方法如下：

```
MediaRecorder  mediaRecorder = new MediaRecorder();
```

2．设置音频采集设备

要进行音频的录制必须使用硬件设备，而在手机开发中一般会使用麦克风作为音频的录制设备（事实上，目前 Android 只支持使用它），设置方法如下：

```
setAudioSource(MediaRecorder.AudioSource.MIC);
```

这里的参数显而易见是指麦克风了。

3．设置音频录制格式

这里所指的格式并不是大家平常所熟知的 MP3、MP4、3GP 等，而是指录制时的编码格式，使用麦克风录制的数据我们称之为裸数据，是未经压缩或任何处理的数据。但很多时候这样的数据不能很好地满足我们的开发需求，因为这真的太大了。所以，一般在开发时我们需要将捕获的数据进行压缩。

而压缩的方法有很多种，编解码本身就是一个很大的课题，在这里的讲解只是抛砖引玉，有兴趣的读者可以继续深入了解。言归正传，Android 的 MediaRecorder 对象支持的压缩方法是 AMR_NB 格式。这也是目前非常流行的一种压缩格式（非常遗憾的是，该对象也仅仅支持这一种压缩格式）。方法如下：

```
setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
```

同样，这里的参数指定压缩格式为 AMR_NB。

4．设置文件输出格式

这里所指的格式就是大家平常所熟知的 MP3、MP4、3GP 等等了。这里就不需笔者花费过多的笔墨了吧。MediaRecorder 对象支持的输出格式包括：RAW_AMR、MPEG_4、THREE_GPP。使用方法如下：

```
setOutputFormat(MediaRecorder. OutputFormat.MPEG_4);
```

括号里的参数可选择上述的 3 种格式之一，这里笔者设置为了 MPEG_4。

5．设置目标文件

各类参数设置完毕后，我们必须告诉 MediaRecorder 对象其录制的音频文件要保存在哪里。这时就需要使用设置目标文件的方法。方法如下：

```
setOutputFile(String path);
```

括号里的参数可以是一个文件的有效对象，或者是一个文件的有效路径。

6．准备录制

现在，准备工作做的已经差不多了。接下来要做的就是告诉 MediaRecorder 对象，我们已经设置完成，需要它准备录制了。这个时候需要调用：

```
Prepare();
```

调用了该方法后，MediaRecorder 对象进入准备就绪状态。接下来就可以进行录制了。

7．开始录制

这个时候 MediaRecorder 对象已经是准备就绪状态，我们需要调用 start()方法通知它开始录制。一旦调用，MediaRecorder 对象就会开始工作，向文件中写入音频数据。方法很简单：

```
Start();
```

调用了该方法后，MediaRecorder 对象进入开始状态，已经在采集数据并按照之前设置的参数向文件中写入数据了。

8．停止录制

当需要停止音频的录制时，只要简单的调用 stop()方法就可以停止 MediaRecorder 对象。方法如下：

```
Stop();
```

调用了该方法后，MediaRecorder 对象进入停止状态，停止采集数据，但并未释放资源。

9．释放资源

当需要停止音频的录制后，其实 MediaRecorder 对象依然占用着资源，要保证系统安全高效地运行，我们需要及时释放该对象占用的资源。方法为：

```
Release();
```

调用了该方法后，MediaRecorder 对象将释放资源，至此一次完整的音频录制结束。

10.1.2　通过实例学习使用 MediaRecoder 录制音频

同样地，本节通过一个实例学习使用 MediaRecorder 类。因为是学习使用音频录制，这里就不贴出效果图了。简单地介绍一下程序，程序本身只有两个按钮：开始按钮——按下后开始录制音频，停止按钮——按下后结束录音。若程序运行成功，读者可以在 SD 卡中找到刚才录制的文件。

1．整体设计

首先关联 xml 布局文件，接着实例化两个按钮，一个命名为 recordBtn，另一个为 stopBtn，分别为其设置监听事件。在 recordBtn 的单击事件中实现录制方法，在 stopBtn 的单击事件中实现停止方法。

```
package com.wes.recoeder;

import java.io.File;                              //导入文件类
import android.media.MediaRecorder;               //导入多媒体录制类
import android.os.Environment;                    //导入环境类
import ...                                        //省略部分类的导入
```

```
public class mainActivity extends Activity {
    MediaRecorder   audioRecorder;
    Button          recordBtn;                            //声明录制按钮
    Button          stopBtn;                              //声明停止按钮
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);

       recordBtn = (Button) findViewById(R.id.button1);    //实例化录制按钮
       stopBtn   = (Button) findViewById(R.id.button2);    //实例化停止按钮

       recordBtn.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                record();                                 //调用录制方法
            }
        });

       stopBtn.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                stop();                                   //调用停止方法
            }
        });
   }
}
```

2. 实现录制方法

相信通过上一小节的讲解，读者阅读以下的代码片段应该没有太大的问题。只要按照上节所讲述的步骤一步步地往下走，读者会发现：一切都很简单。

```
    public void record()
    {
        if (audioRecorder == null)
          {
           audioRecorder = new MediaRecorder();    //实例化 recorder
          }
          String path = Environment.getExternalStorageDirectory().
          getAbsolutePath() + "/test.mp4";
          File file = new File(path);              //根据路径创建文件
          if (file.exists())
              file.delete();                       //若文件存在则删除
           try
           {
              file.createNewFile();                //创建文件
               audioRecorder.setAudioSource(       //设置采集设备为麦克风
                      MediaRecorder.AudioSource.MIC);
               audioRecorder.setOutputFormat(      //设置输出格式为 MPEG 4
                      MediaRecorder.OutputFormat.MPEG_4);
               audioRecorder.setAudioEncoder(      //设置编码格式为 AMR_NB
                      MediaRecorder.AudioEncoder.AMR_NB);
```

```
                audioRecorder.setOutputFile(path);     //设置输出文件路径
                audioRecorder.prepare();          //设置 recorder 状态为准备
                audioRecorder.start();            //设置 recorder 状态为开始
            }
            catch (IOException e)
            {
                e.printStackTrace();                    //捕获 I/O 异常
            }
            catch (IllegalStateException e)
            {
                e.printStackTrace();                    //捕获非法状态异常
            }
            recordBtn.setText("正在录制...");
    }
```

在这里值得一提的是，要注意捕获 IOException 和 IllegalStateException，读者在开始学习使用该类时往往会出现这两个异常。当你遇到时不要急躁，静下心来，仔细找找到底是哪里出了问题。在这样的调试过程中读者会发现自己的编程能力正在逐渐提高！

3. 实现停止方法

当停止按钮按下时，需要停止音频录制。这时要做的事情是停止 recorder 并释放其占用的资源。注意一定要释放资源，否则读者会发现自己的机器越跑越慢甚至出现死机的现象，这就是因为大量内存被浪费了。代码如下：

```
    public void stop()
    {
        if (audioRecorder != null)
        {
            audioRecorder.stop();                      //停止 recorder
            audioRecorder.release();                   //释放 recorder 的资源
            audioRecorder = null;
        }
        recordBtn.setText("录音");
    }
}
```

到这里我们音频录制的例子就结束了，在 SD 卡中读者会找到名为 test.mp4 的文件。使用播放器打开看看，是不是你刚才录制的声音呢？如果一切顺利，那么你现在应该能听到自己录制的声音了。

当然通过系统的播放器播放自己录制的声音文件可能还是不够个性化，也许你觉得还是不够过瘾。那么下一小节笔者将带领读者编写一个自己的 MediaPlayer 播放器。

亲爱的读者你是否理清思路了呢？如果你能够在脑海里清晰地回想起录制音频的 9 个操作步骤，那么不要犹豫，赶紧自己编写一个个性录音机吧！

10.1.3 使用 MediaPlayer 播放音频

上小节读者学会了使用 MediaRecorder 录制音频，可是仅仅录制必然是无法满足开发需求的，更多的时候我们需要播放音频。在 Android 中，一般使用 MediaPlayer 类来播放音频。在这一小节我们将学会使用并深入理解它。

要使用 MediaPlayer 对象需要如下几个步骤：

（1）实例化一个 MediaPlayer 对象。

（2）设置数据源。

（3）准备播放。

（4）开始播放。

（5）停止播放。

（6）释放资源。

接下来，我们来仔细探究每个步骤的具体实现和功能。

1．实例化MediaPlayer对象

通过 new MediaPlayer()方法获得一个 MediaPlayer 对象，以便对其进行设置和操作。方法如下：

```
new MediaPlayer();
```

2．设置数据源

有了 mediaPlayer 对象后，就可以开始播放音频了。那么首先要播放那个文件呢？是了，在播放前我们要设置要播放的目标文件，也就是提供给 MediaPlayer 数据源，方法如下：

```
SetDataSource(filePath);
```

参数可以是一个有效的文件对象或文件的有效路径。

3．准备播放

接下来可能很多读者会想，下一步是不是要设置具体的参数了呢？如文件的压缩格式、读取方式等等。首先，如果你能这么想笔者必须鼓励你。因为你有这样的思维说明你已经养成了一定的编程习惯和思路（作为一个程序员，很多时候我们要敢想敢做，也许你想到的是其他人没有想到的，即使那个人也许编程功底比你高。要知道现在 21 世纪最值钱的是创意!）。但是，笔者同样需要告诉你，使用 mediaPlayer 对象并不需要设置这些参数，这也是它的方便之处。

言归正传，同使用 MediaRecorder 一样，我们需要对其状态进行设置，要开始播放必须先通知其进入准备状态，方法很简单只需调用：

```
Prepare();
```

4．开始播放

准备就绪后我们就可以开始正式播放了，方法为：

```
Start();
```

到这里播放就完成了，但是不要忘记了，播放完成后要停止并释放资源!

5．停止播放

注意内存的使用是编程的一个好习惯，每个类在你可以手动释放资源的时候千万不要

忘记手动释放资源，当然如果无法手动释放资源，那只能指望 GC（垃圾回收）帮你及时清理了。停止播放方法为：

```
Stop();
```

6．释放资源

同 MediaRecorder 一样，最后释放资源：

```
Release();
```

到这里，一个完整的播放流程就结束了。接下来，我们同样通过一个实例来整合刚才的知识。

10.1.4　通过实例学习 MediaPlayer

相信通过上一小节的学习，笔者应该有能力写出一个简易的播放器了，这里笔者就给出一个简单播放器的实例。

1．整体设计

同样界面非常简单，包含一个“播放/暂停”按钮，一个“停止”按钮。初始界面如图 10.1 所示：显示“播放”和“停止”按钮。当按下“播放”按钮后，该按钮会显示“暂停”，如图 10.2 所示。再按下该按钮则暂停播放，界面恢复到图 10.1，当文件播放完毕后也会回到初始界面。



图 10.1　初始界面



图 10.2　播放中界面

好了，功能介绍完后我们一起来实现它吧：

```
package com.wes.test;

import android.media.MediaPlayer;                //导入 MeidaPlayer 类
import android.media.MediaPlayer.OnCompletionListener;     //导入完成监听器
import ...                                       //省略部分导入包

public class mainActivity extends Activity {
    MediaPlayer    audioPlayer;
```

```
    Button          playBtn;                            //声明“播放/暂停”按钮
    Button          stopBtn;                            //声明“停止”按钮
    boolean             isPlaying = false;              //是否正在播放标签
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);

       playBtn = (Button) findViewById(R.id.button1);
                                                        //实例化“播放/暂停”按钮
       stopBtn  = (Button) findViewById(R.id.button2);//实例化“停止”按钮

       playBtn.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                if (isPlaying)
                    pause();                            //调用暂停方法
                else
                {
                    if (audioPlayer == null)
                        play();                         //调用播放方法
                    if (audioPlayer != null)
                        reStart();                      //调用继续播放方法
                }
            }
        });

       stopBtn.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                stop();                                 //调用停止方法
            }
        });
   }
}
```

通过后面的注释，理解这块整体设计应该不难。

2. 播放功能实现

播放功能是该应用的主要功能，我们就来仔细看这段代码并回忆上小节中讲的几个步骤：

```
public void play()
{
    if (audioPlayer == null)
        audioPlayer = new MediaPlayer();                    //实例化播放器
    String path = Environment.getExternalStorageDirectory().
    getAbsolutePath() + "/test.mp4";
    try
        {
        audioPlayer.setDataSource(path);                    //设置数据源
        audioPlayer.prepare();                              //设置状态为准备好
```

```
        audioPlayer.start();                                //设置状态为开始
        audioPlayer.setOnCompletionListener(new OnCompletionListener()
        {
            @Override
            public void onCompletion(MediaPlayer arg0)
            {
                stop();                  //监听是否完成播放，若是则调用停止方法
            }
        });
    }
    catch (IllegalArgumentException e)
    {
        e.printStackTrace();                                //捕获非法参数异常
    }
    catch (IllegalStateException e)
    {
        e.printStackTrace();                                //捕获非法状态异常
    }
    catch (IOException e)
    {
        e.printStackTrace();                                //捕获 I/O 异常
    }
    playBtn.setText("暂停");
    isPlaying = true;
}
```

注意 audioPlayer.setOnCompletionListener();，该语句实现了监听文件是否播放完全，在本应用中播放完全后就将播放器置为停止状态。当然很多时候你可以实现一些其他的操作，比如播放完后跳转到另一界面等。

3．暂停功能实现

在播放时，我们经常可能被一些突如其来的事情打断，这时我们就需要将播放器暂停。方法非常简单，如下所示：

```
public void pause()
{
    if (audioPlayer != null)
    {
        audioPlayer.pause();                                //设置状态为暂停
    }
    isPlaying = false;
    playBtn.setText("播放");
}
```

4．重新播放功能实现

暂停了之后我们会重新开始播放，这个时候只需重新调用 start()方法就可以了：

```
public void reStart()
{
    audioPlayer.start();                                    //重新开始播放
    isPlaying = true;
    playBtn.setText("暂停");
}
```

5．停止功能实现

当我们不需要使用播放器时，我们就停止播放器并释放其占用的资源以供其他对象使用：

```
    public void stop()
    {
     if (audioPlayer != null)
     {
         audioPlayer.stop();                          //停止播放
         audioPlayer.release();                       //释放资源
         audioPlayer = null;
     }
     playBtn.setText("播放");
}
```

到这里我们就已经学会了音频的录制和播放，但很多时候是不能满足开发需求的，如语音通讯。这个时候就需要使用另外两个类（下节讲解）来实现音频的录制和播放。可能这两个类的使用会比 MediaRecorder 和 MediaPlayer 麻烦一些，但是相信只要读者保持高昂的斗志和平静的心态，任何困难都不能阻止我们前进的脚步。

10.2　深度处理音频

上一节中我们学习了简单处理音频的方法，这一节将学习使用一个更加“有深度”的类来操作音频数据。之前学习的方法只需获得一个 mediarecorder 对象并设置好参数就可以开始录制了，虽然很简单但是不能直接看到我们录制的音频数据，这未免不是一种遗憾。在这一节中我们学习使用 AudioRecord 类和 AudioPlayer 类来直接对音频数据进行操作。

10.2.1　使用 AudioRecod 录制音频

在 Android 音频系统中，AudioRecord 类较 MeidaRecorder 类更底层，所以封装的相对较少，可供操作的空间更大。经过了上一节的学习很多读者会想，通过之前的方法的确可以录制音频数据，但是怎样将获得的数据展现或传递呢？是不是只能传递文件呢？事实上，上述问题通过使用 AudioRecord 类就可以迎刃而解。接下来就来进入 AudioRecord 类的学习。

使用 AudioRecord 步骤相对简单，不需要进行各种设置，因为在获得其对象的时候已经传递了相关参数。具体步骤为：

（1）获得 AudioRecord 对象。

（2）开始录音。

（3）获得音频数据。

（4）停止录制。

（5）释放资源。

乍一看似乎步骤比使用 MediaRecorder 简单，但是里面的内容可不少哦。接下来就仔

细探究各个步骤的具体做法。

1. 获得AudioRecord对象

```
AudioRecord audioRecord = new AudioRecord(int audioSource, int sampleRateInHz, int channelConfig, int audioFormat, int bufferSizeInBytes);
```

新建 AudioRecord 对象时需要 5 个参数：

（1）audioSource：录制使用的资源，这里是麦克风，即 MediaRecorder.AudioSource.MIC。

（2）sampleRateInHz：每秒的采样率，意义为每秒采集多少次样本，单位是 Hz。设置为一个整数值，一般为 8000，或者 11400 等，如果读者有兴趣可以深入研究。

（3）channelConfig：声道配置，也就是平时所说的单声道，双声道。常用的参数为 AudioFormat.CHANNEL_CONFIGURATION_MONO（单声道），或 AudioFormat.CHANNEL_CONFIGURATION_STEREO（双声道，立体声）。

（4）audioFormat：编码方式，即每次采样的位数，可以设置为 AudioFormat.ENCODING_PCM_16BIT（16 位采样），或 AudioFormat.ENCODING_PCM_8BIT（8 位采样）。

（5）bufferSizeInBytes：为 AudioRecord 开辟的缓存区大小，以 byte 为单位。

2. 开始录音

获得了一个 AudioRecord 对象后就可以开始录音了，因为我们已经将参数设置好了，方法是：

```
AudioRecord.startRecording();
```

3. 获得音频数据

开始录制后，我们就可以从麦克风读取采集到的数据了，这也是和 MediaRecorder 最大的不同，这里我们可以看到以 byte[]数组形式组成的音频数据。方法为：

```
AudioRecord.read(short[] audioData, int offsetInShorts, int sizeInShorts);
```

这里有 3 个参数：

（1）audioData，用来存放音频数据的 byte[]数组或是 short[]数组。

（2）offsetInShorts 或 offsetInBytes：在数组中的偏移量，如果从头开始录则设置为 0。

（3）sizeInShorts 或 sizeInBytes：数组的大小。

这样，每次 audioRecorder 就从麦克风读取 sizeInShorts 大小的数据写入到 audioData 中，我们只需将 audioData 写入到文件中就可以保存录音了，或者将其通过 socket 在网络上传输，就可以达到实时通讯的目的。

4. 停止录制

与 MediaRecorder 一样，方法为：

```
AudioRecord.stop()
```

5. 释放资源

同样调用方法：

```
AudioRecord.release()
```

好的，讲解到这里相信读者应该对 AudioRecorder 有一个初步的了解了，接下来就通过实例来实战一下！

10.2.2　通过实例学习使用 AudioRecod 录制音频

本实例使用的布局文件与上一节中的 MediaRecorder 相同，不同的是其中录制功能的实现，接下来是整体设计。

1. 整体设计

这里的整体设计同样非常简单，只是导入的类的作用大家要做到心中有数，只有从整体上能够把握一个程序，阅读起代码来才能从容不迫、游刃有余：

```
package com.wes.audio;

import java.io.BufferedOutputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;                    //到这里为导入文件读写相关的类
import java.text.DateFormat;
import java.text.SimpleDateFormat;              //导入日期格式类
import java.util.Calendar;                      //导入日历类
import java.util.Date;                          //导入日期类
import android.media.AudioFormat;               //导入音频格式类
import android.media.AudioRecord;               //导入音频录制类
import android.media.MediaRecorder;             //导入媒体录制类
import ...                              //此处省略部分导入类

public class mainActivity extends Activity {
    MediaRecorder   audioRecorder;
    Button          recordBtn;                  //声明“录制”按钮
    Button          stopBtn;                    //声明“停止”按钮
    boolean         isRecording = false;        //正在录音标签
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);

       recordBtn = (Button) findViewById(R.id.button1);//实例化“录制”按钮
       stopBtn   = (Button) findViewById(R.id.button2);//实例化“停止”按钮

       recordBtn.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                doRecord();
            }
        });

       stopBtn.setOnClickListener(new View.OnClickListener()
```

```
        {
            @Override
            public void onClick(View arg0)
            {
                stop();                                         //调用停止方法
            }
        });
}
}
```

框架搭建的非常简单，这里笔者就不再赘述，接下来看功能实现。

2．录制功能实现

这里按照前一小节的讲解，按部就班就可以了：

```
public  void  record()
{
    int frequency = 8000;                               //每秒 8000 采样
    String filePath = Environment.getExternalStorageDirectory().
getAbsolutePath() + "/" + getFileTime() + ".pcm";
    File file = new File(filePath);                     //创建文件对象

    try
      {
        file.createNewFile();                           //新建文件
        OutputStream os = new FileOutputStream(file);
        BufferedOutputStream bos = new BufferedOutputStream(os);
        DataOutputStream dos = new DataOutputStream(bos);   //获得流操作对象

        int bufferSize = AudioRecord.getMinBufferSize(8000,
                                                //获得 recorder 的最小内存
            AudioFormat.CHANNEL CONFIGURATION MONO,
            AudioFormat.ENCODING PCM 16BIT);

        AudioRecord audioRecord = new AudioRecord(
                        MediaRecorder.AudioSource.MIC,
                                //获得 AudioRecord 对象
                        frequency,      //每秒 8000 采样
                        AudioFormat.CHANNEL_CONFIGURATION_MONO,
                                //单声道录制
                        AudioFormat.ENCODING_PCM_16BIT,
                                //PCM16 位编码方式
                        bufferSize);
        short[] buffer = new short[bufferSize];
        audioRecord.startRecording();   //开始录音
        isRecording = true;

        while (isRecording)
        {
         int bufferReadResult = audioRecord.read(buffer, 0, bufferSize);
                                //采集音频数据
         for (int i = 0; i < bufferReadResult; i++)
         {
             dos.writeShort(buffer[i]);//写入文件
         }
        }
        audioRecord.stop();             //停止录制
        audioRecord.release();          //释放资源
```

```
        dos.close();                          //关闭流
              }
     catch (Exception e)
     {
         e.printStackTrace();
     }
}
```

需要注意的是两点：

（1）这里使用了：

```
AudioRecord.getMinBufferSize(8000,AudioFormat.CHANNEL_CONFIGURATION_MON
O, AudioFormat.ENCODING_PCM_16BIT);
```

方法获得了 AudioRecord 需要的最小内存，具体的计算方法我们不需要探究，需要知道的是我们设置的参数值必须大于这个最小的缓存大小，否则程序出现异常。

（2）这里使用了 getFileTime()方法得到当前时间作为文件名，具体的实现方法会在后面介绍。

3．实现停止功能

接下来实现“停止”按钮的单击事件，注意只需将正在录制标签置为 false，则 record()跳出 while 循环，执行语句：

```
audioRecord.stop();                                    //停止录制
         audioRecord.release();                        //释放资源
         dos.close();                                  //关闭流
```

从而实现 audioRecord 的关闭及释放。

```
    public void stop()
    {
        isRecording = false;
                                                       //设置录制标签为 false
        recordBtn.setText("录音");

    }
```

4．获得文件名方法实现

这里使用了几个 Java 自带的类，读者可以通过笔者这个小例子学习其简单使用，如果有兴趣可以自行深入了解。

```
public String getFileTime()
{
    Date curTime = Calendar.getInstance().getTime();        //获得当前时间
    DateFormat df = new SimpleDateFormat("yyyyMMddHHmmss");
                                                            //设置输出形式
    String time  = df.format(curTime).toString();           //得到时间
    return time;
}
```

5．开辟录制线程

可能写到第 4 步很多读者觉得这个实例应该结束了，其实不然，当执行 record 方法的时候会将主线程阻塞，导致程序异常。因为线程执行了一个死循环，即不停地读取数据。所以我们要另外新建一个线程，并设置它的 run 方法为录音，方法如下：

```
public  void  doRecord()
{
    Thread audioThread = new Thread(new Runnable(){        //创建录音线程
                public void run()
                {
                    record();                              //调用录制方法
                }
            });
            audioThread.start();
            recordBtn.setText("录音中...");
}
```

注意新建完线程后，必须跟上 thread.start()以开启线程，否则线程将不会启动。到这里有一些读者又会产生疑问，开启了线程执行那程序是不是要另外写一个方法结束这个线程呢？

这里要告诉大家的是，手动强制结束线程的方法已经不用了，现在的做法是通过执行 run()方法中的代码来结束线程。也就是说，run()结束了则线程就自动关闭了。所以在之前的 stop()方法中结束了 record()方法的同时也“顺便”将 audioThread 线程一起给结束了。讲到这里相信大家应该会使用 AudioRecord 类了吧？那么保存的文件我们怎样播放呢？不要着急，下一小节再为大家一一道来。

10.2.3　使用 AudioTrack 播放音频

学习完使用 AudioRecord 录制音频后，继续学习与其对应的播放类——AudioTrack。AudioTrack 播放的必须是未经压缩的数据也就是“裸数据”，或称为原始数据。例如，上一小节中从麦克风处 Read 到 short 或者 byte 数组，这就是原始的音频数据。这类数据经过各种各样的压缩形式后就可以得到一些比如 MP3、MP4、3GP 等格式的文件。细心的读者会发现上一小节中我们将读取到的裸数据保存为了一个文件，该文件的后缀名为.pcm。

那何为 PCM 文件呢？PCM（pulsecodemodulation，中文称作脉冲编码调制）。这里涉及的知识又需要读者自己去深入挖掘，在这里只能抛砖引玉。宽泛地理解，PCM 数据就是数字信号对模拟信号的一个抽样量化和编码。读者可以直接理解为推送给硬件的最基础的数据。

使用 AudioTrack 步骤也非常简单，具体步骤为：

（1）获得 AudioTrack 对象。

（2）开始播放。

（3）写入音频数据。

（4）停止录制。

（5）释放资源。

读者可以发现其步骤与 AudioRecord 一一对应。接下来就仔细探究各个步骤的具体做法。

1. 获得AudioTrack对象

在新建 AudioTrack 对象的时候同样需要传递若干参数，让我们先来看方法：

```
AudioTrack.AudioTrack(int   streamType,   int   sampleRateInHz,   int
channelConfig, int audioFormat, int bufferSizeInBytes, int mode);
```

新建 AudioTrack 对象时需要 6 个参数：

（1）streamType：播放流的类型，一般设置为 AudioManager.STREAM_MUSIC ，即音乐类型。可供选择的参数还包括：

- AudioManager.STREAM_ALARM（报警类型）。
- AudioManager.STREAM_DTMF（双音多频类型 dual-tone multifrequency）。
- AudioManager.STREAM_NOTIFICATION（消息类型）。
- AudioManager.STREAM_RING（铃声类型）。
- AudioManager.STREAM_SYSTEM（系统类型）。
- AudioManager.STREAM_VOICE_CALL（电话类型）。

当然设置这些参数可能对用户来说意义不大，但是通过这些参数，系统可以很好地管理音频系统。例如，你在听音乐，此时为 MUSIC 模式，这个时候进来一个电话，那系统肯定要打断 MUSIC 接入 VOICE_CALL。当你接听电话的时候你又觉得声音太小，此时调节的就是通话音量。当通话结束重新回到 MUSIC 状态时，此时的音乐音量应该还是开始通话前的音量。

这就是设置 TYPE 参数的好处了。

（2）sampleRateInHz：每秒的采样率，意义为每秒采集多少次样本，单位是 Hz，设置为一个整数值，一般为 8000，或者 11400 等如果读者有兴趣可以深入研究。

（3）channelConfig：声道配置，也就是平时所说的单声道，双声道。常用的参数为：AudioFormat.CHANNEL_CONFIGURATION_MONO（单声道）或 AudioFormat.CHANNEL_CONFIGURATION_STEREO（双声道，立体声）。

（4）audioFormat：编码方式：即每次采样的位数，可以设置为：AudioFormat.ENCODING_PCM_16BIT 16 位采样，或者是 AudioFormat.ENCODING_ PCM_8BIT　8 位采样。

（5）bufferSizeInBytes：为 AudioRecord 开辟的缓存区大小，以 byte 为单位。

（6）mode：模式，一般设置为 AudioTrack.MODE_STREAM，或者设置为 AudioTrack.MODE_STATIC。这里的两个参数用户会感受强烈一些，设置为 STREAM 模式时，读者可以通过流的形式不停地向 Track 中添加数据，而 AudioTrack 会负责播放这些数据，其工作方式与 Socket 类似。一个实时通话的程序就需要这种模式，将从 Socket 端读取的字节流传递到 AudioTrack 中就完成了实时播放了。

使用 AudioTrack.MODE_STATIC 模式时，AudioTrack 不会从流中不停地读，而是从一块预先开辟的 Buffer 中读取数据并播放。与 AudioTrack.MODE_STREAM 相比，好处是会减少很多消耗（因为 AudioTrack.MODE_STREAM 模式下，Java 需要不停地调用 Native 方法），缺点是不够灵活。

2．开始播放

获得了一个 AudioTrack 对象后就可以开始播放了，方法是：

```
AudioTrack.play()
```

3．写入音频数据

与 AudioRecord 类相对应的，开始播放后就需要向 AudioTrack 中写入数据。方法为：

```
.AudioTrack.write(short[] audioData, int offsetInShorts, int sizeInShorts)
```

这里同样有 3 个参数：

（1）audioData，用来存放音频数据的 byte[]数组或 short[]数组。

（2）offsetInShorts 或 offsetInBytes：在数组中的偏移量，如果从头开始录则设置为 0。

（3）sizeInShorts 或 sizeInBytes：数组的大小。

这样，每次由 offsetInShorts 开始从 audioData 中读取 sizeInShorts 个数据，添加到 AudioTrack 中以供其播放。

4．停止播放

与 AudioRecord 类似，方法为：

```
AudioTrack.stop()
```

5．释放资源

同样调用方法：

```
AudioTrack.release()
```

相信到这里很多读者应该对音频的播放有一个清楚的认识了。到目前为止我们一共学习了 4 个类分别进行音频的录制和播放，只要熟练掌握这 4 个类，一般情况下做音频的应用开发已经足够了，那么接下来依旧通过一个实例来操练操练吧。

10.2.4　通过实例学习使用 AudioTrack 录制音频

这一小节我们使用 AudioTrack 类来编写一个播放 PCM 文件的简单播放器，在开始这个播放器之前我先来构思一下这个播放器：两个按钮，一个“播放”，一个“停止”。按下“播放”按钮则新建 AudioTrack 对象，设置为 play 状态，接着从文件读取数据到数组中，最后将数据写入 AudioTrack 中，完成播放。按下“停止”按钮则停止播放器并释放资源。一切似乎都很简单。但是，亲爱的读者们不要忘记使用线程来进行播放！因为使用 AudioTrack 播放同样是阻塞的，在主线程中执行极易引起异常。好了，分析到这里让我们赶紧开始我们的编程之旅吧。

1．整体设计

在本节开始我们已经大概完成了整体的设计，这里笔者就不再赘述，让我们来分析一下代码吧：

```
package com.wes.audio;

import android.media.AudioManager;                    //导入音频管理类
import android.media.AudioTrack;                      //导入 AudioTrack 类
import ...                                            //此处省略部分导入包

public class mainActivity extends Activity {
    AudioTrack      audioTrack;                       //声明 audioTrack 对象
    Button          playBtn;                          //声明“播放”按钮
    Button          stopBtn;                          //声明“停止”按钮
    boolean             isPlaying = false;            //是否正在播放标签

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);

       playBtn = (Button) findViewById(R.id.button1);  //实例化“播放”按钮
       stopBtn   = (Button) findViewById(R.id.button2);//实例化“停止”按钮

       playBtn.setOnClickListener(new View.OnClickListener()
        {
           @Override
           public void onClick(View arg0)
           {
               playBtn.setText("播放中...");
               isPlaying = true;                      //设置正在播放标签为 true
               Thread thread = new Thread(new Runnable()  //开辟播放线程
               {
                   @Override
                   public void run()
                   {
                       if (audioTrack == null)
                           play();                            //调用播放方法
                   }
               });
               thread.start();                                //开始线程
           }
        });

       stopBtn.setOnClickListener(new View.OnClickListener()
        {
           @Override
           public void onClick(View arg0)
           {
               stop();                                        //调用停止方法
           }
        });
   }
 }
```

现在再来看这部分整体设计是不是感觉智珠在握呢？笔者在这里一再强调整体感、大局观，是一个优秀的程序员必须具备的素质。只有拥有了一个广阔的视野，才不会把自己陷入编程的泥淖，以一种超然的目光看工程，一切都很清晰，一切是如此的简单。

2. 实现播放功能

结合上小节的各个步骤的详细讲解，再来看这部分的代码会有事半功倍的效果。这里笔者的文件是使用上一小节编写的 AudioRecord 录制的文件，由于使用了时间点作为文件名，所以每个读者基本上都会不同，读者在尝试编程的时候切勿照搬。

```
public void play()
{
while(isPlaying)
    {
     String path =Environment.getExternalStorageDirectory().
     getAbsolutePath()+ "/20110904195915.pcm";
     File file = new File(path);
     int musicLength = (int)file.length();
     short[] music = new short[musicLength];
     if (audioTrack == null)
        audioTrack = new AudioTrack(AudioManager.STREAM_MUSIC,
                                                       //实例化播放器
                    8000,                              //采样率 8000
                    AudioFormat.CHANNEL_CONFIGURATION_MONO,    //单声道
                    AudioFormat.ENCODING_PCM_16BIT,    //每次采样 16 位
                    musicLength,                       //内存大小为 musicLength
                    AudioTrack.MODE_STREAM);           //设置为流模式
     audioTrack.play();                                //设置状态为播放

         try
         {
             InputStream is = new FileInputStream(file);
             BufferedInputStream bis = new BufferedInputStream(is);
             DataInputStream dis = new DataInputStream(bis);//获得数据流
             int i = 0;
             while (dis.available() > 0)       //条件为如果文件中还有可读数据
             {
        music[i++] = dis.readShort();          //将数据保存到数组中
        }
             audioTrack.write(music, 0, musicLength);
                                               //向 AudioTrack 中写入数据
         }
         catch (FileNotFoundException e)
         {
             e.printStackTrace();
         }
         catch (IOException e)
         {
             e.printStackTrace();
         }
      }
}
```

完成这部分主题功能后就剩下简单的结束了。代码虽然简单，但重要性依旧，依然是那句话，良好的习惯是一点一滴养成的。

3. 实现停止功能

这里我就不再赘述了，读者对这部分代码应该很熟悉了。

```
Public void stop()
{
    if (audioTrack != null)
    {
        isPlaying = false;                           //结束线程
        audioTrack.stop();                           //停止播放
        audioTrack.release();                        //释放资源
        audioTrack = null;
    }
    playBtn.setText(“播放”);
}
```

好啦，到这里我们的音频学习就告一段落，有句话叫师傅领进门，修行靠个人。例如，在游戏开发中经常使用的 SoundPool 类，播放一些短促的音效效果非常好，读者如果对音频感兴趣可以继续钻研。

10.3　学 会 拍 照

本章的前两节学习了音频的相关处理，接下来将学习摄像头的使用。摄像头大体上有两个功能：拍照以及录制视频。本节我们就学习使用它的第一个功能——拍照。拍照是摄像头的基本使用，我们将通过 Android 提供的 Camera 类完成静态图像的捕获。

10.3.1　通过 Camera 类完成拍照

Android SDK 提供了 android.hardware.Camera 类对摄像头进行操作，使用它，开发人员可以很方便地完成拍照的功能。完成一个拍照功能我们需要完成两个部分：第一个部分是预览，第二个部分就是拍照了。首先我们分析第一部分预览。

1. 预览功能

相信大家都使用过手机的拍摄功能，拍摄肯定不能是盲目地拍。如果没有了预览，摄像就是“赌人品”，那就没有摄影家这个称呼了。在 Android 中完成预览功能，我们可以使用：

```
Camera.setPreviewDisplay(SurfaceHolder holder)
```

这个方法中的参数大家可能还不是很熟悉，其实我们可以将其看作是一个 SurfaceView 的句柄。也就是说通过 SurfaceHolder，我们可以操作 SurfaceView 的大小、格式等等。那么读者也许又产生了一个新的疑问，什么是 SurfaceView 呢？

SurfaceView 是视图（View）的继承类，视图里内嵌了一个专门用于绘制的 Surface。你可以控制这个 Surface 的格式和尺寸，Surfaceview 控制这个 Surface 在屏幕上的绘制位置。

由此可见，我们使用 SurfaceView 在屏幕上渲染出一片区域，然后将摄像头捕捉到的画面显示在该区域内。完成一个 SurfaceView 的设置需要经过以下 5 个步骤：

（1）创建一个 SurfaceView。

（2）获得操作对象。

（3）获得 SurfaceHolder 对象。

（4）实现 SurfaceHolder.Callback 接口，并传递给 SurfaceHolder。

（5）设置 SurfaceHolder 的类型。

接下来我们进行各个步骤的详解。

（1）在 xml 代码中创建一个 SurfaceView。例如，创建一个 320×240 的 SurfaceView 语法格式如下：

```
<SurfaceView
        android:id="@+id/surfaceView1"
        android:layout_width="320px"
        android:layout_height="240px"
        />
```

（2）在 Java 代码中获得其操作对象：

```
SurfaceView  sv = (SurfaceView)findViewById(R.id.surfaceView1);
```

（3）获得 SurfaceHolder 对象：

```
holder = sv.getHolder();
```

（4）为 SurfaceHolder 添加回调接口。

前文提到过 SurfaceHolder 作为一个 SurfaceView 的句柄而存在，那么它必须时刻关心 SurfaceView 的状态，何时创建、何时改变、何时销毁。而这些在 SurfaceHolder 接口中都可以被实现。实现该接口需要完成 3 个函数，分别是：

```
@Override
  public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int
arg3)
  {
  }

  @Override
  public void surfaceCreated(SurfaceHolder arg0)
  {
  }

  @Override
  public void surfaceDestroyed(SurfaceHolder arg0)
  {
  }
```

从字面上就可以看出其具体意义了，SurfaceCreated()在 SurfaceView 被创建时回调，SurfaceChanged()在 SurfaceView 被改变时使用，如发生横竖屏变化时，SurfaveDestroyed()在 SurfaceView 被销毁时使用，如将其设置为不可见时。

实现了该接口后我们就可以将其作为一个参数传递给 SurfaceHolder，使用方法为：

```
SurfaceHolder.addCallback(Callback arg0)
```

（5）为 SurfaceHolder 设置类型：

```
SurfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

这里的参数设置为 SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS，表示该 Surface 不包含原生的数据，它的数据由其他对象提供。还可以使用的参数有：

- ❑ SURFACE_TYPE_NORMAL：普通类型，通过 RAM 缓存原生数据。
- ❑ SURFACE_TYPE_HARDWARE：硬件类型，适用于 DMA（Direct memory access）引擎和硬件加速。
- ❑ SURFACE_TYPE_GPU：图形处理器（Graphic Processing Unit）类型，适用于 GPU 加速。

2．配置Camera的预览参数

经过以上 5 个步骤，我们成功得到了一个可以操作的 SurfaceView。接下来要做的工作是配置 Camera 的预览参数，需要以下 5 个步骤：

（1）获得 Camera 对象。

（2）获得相机参数。

（3）设置预览大小。

（4）绑定 SurfaceView。

（5）开始预览。

接下来分析各个步骤的语法。

（1）获得 Camera 操作对象：

```
Camera camera = Camera.open()
```

（2）获得相机的参数，以便设置：

```
Camera.Parameters param = camera.getParameters();
```

（3）设置预览大小，使用方法，参数为宽度和高度：

```
Parameters.setPreviewSize(int width, int height)
```

（4）将 Camera 对象与 SurfaceView 绑定：

```
Camera.setPreviewDisplay(SurfaceHolder holder)
```

（5）开始预览，方法为：

```
Camera.startPreview()
```

经过以上 5 个步骤，我们成功地将预览功能完成，接下来我们需要完成的就是拍照功能了。

3．拍摄功能

拍照功能较之预览功能反而要显得简单一些，其主要的方法为：

```
Camera.takePicture(ShutterCallback shutter,
PictureCallback raw,
PictureCallback jpeg)
```

这里有 3 个参数，细心的读者仔细看它们的命名，就会发现其实这 3 个参数是 3 个接口的实现。所以顺理成章地，我们需要完成拍照功能就需要以下 4 个步骤：

（1）实现 ShutterCallback 接口，该接口在关闭快门时被回调：

```
private ShutterCallback shutter = new ShutterCallback();
```

（2）实现 PictureCallback raw 接口，该接口在获得未经压缩的照片时被调用：

```
private PictureCallback raw = new PictureCallback();
```

（3）实现 PictureCallback jpeg 接口，该接口与第二步类似，不同的是它在获得 JPEG 格式的照片时被调用，我们就重点讲解该接口：

```
private PictureCallback jpeg = new PictureCallback()
```

在该接口中需要实现方法：

```
public void onPictureTaken(byte[] bytes, Camera _camera)
```

这里第一个参数就是照片的原始数据了，我们可以将其转换为位图，语法如下：

```
Bitmap bm = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
```

拥有了位图的操作对象，我们就可以将其保存为文件了，语法为：

```
Bitmap.compress(CompressFormat format, int quality, OutputStream stream)
```

这里的 3 个参数，第一个是压缩格式，可以将其设置为 Bitmap.CompressFormat.JPEG；第二个参数是压缩质量，一般设为 80、100 等；第三个参数是文件的输出流，通过输出流将数据保存为文件。

（4）完成拍摄函数，将这 3 个接口传递给该函数：

```
Camera.takePicture(ShutterCallback shutter, PictureCallback raw,
PictureCallback jpeg)
```

通过以上 4 个步骤，我们完成了简单的拍摄功能，照片会被保存为 JPEG 格式的图片。接下来我们依然通过一个实例完成一个简单的摄像机。

10.3.2　实例——简易摄像机

一个简单的摄像机包括 3 个部分：第一部分是预览，在拍摄时将摄像头捕获的内容传递给 SurfaceView；第二部分是照相，按下拍摄键，我们希望能拍摄照片并保存为 JPEG 格式的图片；第三部分显示，将拍摄的图片显示在 ImageView 上。

首先看布局文件——xml 代码。

布局文件中，我们需要一个 SurfaceView 用于预览，一个 ImageView 用于展示图片，两个 Button 分别用来触发预览和拍摄功能。代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
   <LinearLayout
   android:orientation="vertical"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   >
       <SurfaceView
```

```
        android:id="@+id/surfaceView1"
        android:layout_width="320px"
        android:layout_height="240px"
        />
        <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="preview"
        android:textSize="18sp"
        />
        <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="take"
        android:textSize="18sp"
        />
    </LinearLayout>
    <ImageView
     android:id="@+id/imageView1"
    android:layout_width="320px"
    android:layout_height="240px"
    android:layout_marginLeft="30px"
    />
</LinearLayout>
```

在 Java 代码中，首先完成整体设计：

```
package com.wes.demo;

import ...                          //省略部分导入
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.PixelFormat;
import android.hardware.Camera;
import android.hardware.Camera.AutoFocusCallback;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class MediaDemo extends Activity implements SurfaceHolder.Callback
{
    SurfaceView         sv;
    SurfaceHolder       holder;
    Button              preBtn;
    Button              takeBtn;
    ImageView           iv;
    Camera              camera;
    String              filePath = "/sdcard/camera.jpg";    //文件保存路径

    @Override
public void onCreate(Bundle savedInstanceState)
 {
       super.onCreate(savedInstanceState);
       requestWindowFeature(Window.FEATURE_NO_TITLE);
       setContentView(R.layout.main);
       setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_
       LANDSCAPE);
       initView();                                            //初始化界面
```

```
        preBtn.setOnClickListener(new OnClickListener()
         {
             @Override
             public void onClick(View arg0)
             {
                 preview();                          //开始预览
             }
         });

    takeBtn.setOnClickListener(new OnClickListener()
         {

             @Override
             public void onClick(View arg0)
             {
                 takePicture();                      //完成拍摄
             }
         });
}
}
```

注意加粗部分的代码，如果没有该代码，预览和屏幕的方向会不一致，看上去会非常地别扭。读者可以尝试着注释掉本行代码，重新运行一下程序看看效果如何。接下来需要实现 SurfaceHolder.Callback 接口：

```
@Override
    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int
arg3)
    {
        // TODO Auto-generated method stub
        camera.release();                       //释放资源

    }

    @Override
    public void surfaceCreated(SurfaceHolder arg0)
    {
        // TODO Auto-generated method stub

    }

    @Override
    public void surfaceDestroyed(SurfaceHolder arg0)
    {
        // TODO Auto-generated method stub
        if (camera != null)
        {
            camera.release();               //销毁时释放资源
            camera = null;
        }
    }
```

接下来完成初始化界面的方法，注意这里的 SurfaceView 和 SurfaceHolder 的作用和关系，体会各个回调函数的被调用时机。通过这里的实际代码，再回顾前文所讲的各个步骤详解，可以达到举一反三的效果：

```
private void initView()
    {
```

```
    sv = (SurfaceView)findViewById(R.id.surfaceView1);
    iv = (ImageView)findViewById(R.id.imageView1);
    preBtn = (Button)findViewById(R.id.button1);
    takeBtn = (Button)findViewById(R.id.button2);

    holder = sv.getHolder();                                   //获得 holder 对象
    holder.addCallback(this);                                  //添加回调函数接口
    holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);   //设置类型
    }
```

接下来的部分是预览部分，这里需要注意：设置的一些值必须是 Camera 支持的，否则会出现异常，我们可以通过粗体部分获得支持的一些参数，从而避免胡乱的尝试：

```
   private void preview()
   {
    camera = Camera.open();                          //打开相机，获得相机对象
    try
        {
            Camera.Parameters param = camera.getParameters();
            //获得支持的预览大小，设置不支持的大小会出现异常
            List<Camera.Size> preSizes = param.getSupportedPreviewSizes();
            for(Camera.Size s :preSizes)
            {
                Log.i("SupportedPreviewSize:",s.height + "x" +s.width);
            }
            //获得支持的照片大小，设置不支持的大小会出现异常
            List<Camera.Size> picSizes = param.getSupportedPreviewSizes();
            for(Camera.Size s :picSizes)
            {
                Log.i("SupportedPictureSize:",s.height + "x" +s.width);
            }
            param.setPreviewSize(320, 240);              //设置预览大小
            param.setPictureFormat(PixelFormat.JPEG);    //设置照片格式
            param.setPictureSize(512, 384);              //设置照片大小
            camera.setParameters(param);    //将参数传递给相机对象
            camera.setPreviewDisplay(holder);
                                            //使相机预览显示在 SurfaceView 上
            camera.startPreview();          //开始预览
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
}
```

这里的步骤应当是相当清晰的，如果仍然存在疑问不妨再翻看一下上一节。最后完成拍摄功能，注意 3 个接口的实现，至于 takePicture()函数的调用反而不那么重要：

```
   //关闭的接口，作为一个参数传递给 takePicture 函数
   private ShutterCallback shutter = new ShutterCallback()
   {
        @Override
        public void onShutter()
        {
            //关闭快门
        }
   };
  //未压缩的照片处理接口，作为一个参数传递给 takePicture 函数
```

```
private PictureCallback raw = new PictureCallback()
{

    @Override
    public void onPictureTaken(byte[] bytes, Camera  camera)
    {
        //RawPicture
    }

};
//JPG 格式的照片处理接口，作为一个参数传递给 takePicture 函数
private PictureCallback jpeg = new PictureCallback()
{
    @Override
    public void onPictureTaken(byte[] bytes, Camera  camera)
    {
        // JPG Picture,第一个参数就是照片的字节数组数据，强转为位图类型
        Bitmap bm = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
        File file = new File(filePath);
        try
        {
            FileOutputStream fos = new FileOutputStream(file);
            BufferedOutputStream bos = new BufferedOutputStream(fos);
            //采用压缩转档的方法，保存为 JPEG 格式
            bm.compress(Bitmap.CompressFormat.JPEG, 80, bos);
            bos.flush();
            bos.close();
            iv.setImageBitmap(bm);

            camera.stopPreview();
            camera.release();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
};
//自动对焦的接口，对焦完毕则拍照
private AutoFocusCallback afc = new Camera.AutoFocusCallback()
{
    @Override
    public void onAutoFocus(boolean focused, Camera arg1)
    {
        if (focused)
        {
            takePicture();
        }
    }
};

private void takePicture()
{
    camera.takePicture(shutter, raw, jpeg);
}
```

这里实现了全部的 3 个接口，只是希望读者能够对这些接口有一个更感性的认识。实际上如果你仅仅需要完成拍摄 JPEG 格式的图片的话，就没有必要实现 ShutterCallback 接口和 PictureCallback raw 接口，当调用 takePicture()方法时以 null 代替就可以了。例如，我

们可以将拍照方法修改为：

```
private void takePicture()
 {
    camera.takePicture(null, null, jpeg);
 }
```

代码编写到这里就全部完成了。运行一下看看效果吧，因为模拟器是无法真正拍照的，所以我们必须进行真机测试。拍摄后的效果如图 10.3 所示，左边的图片是 SurfaceView 预览图片，右边是 ImageView 显示拍摄成功的图片。

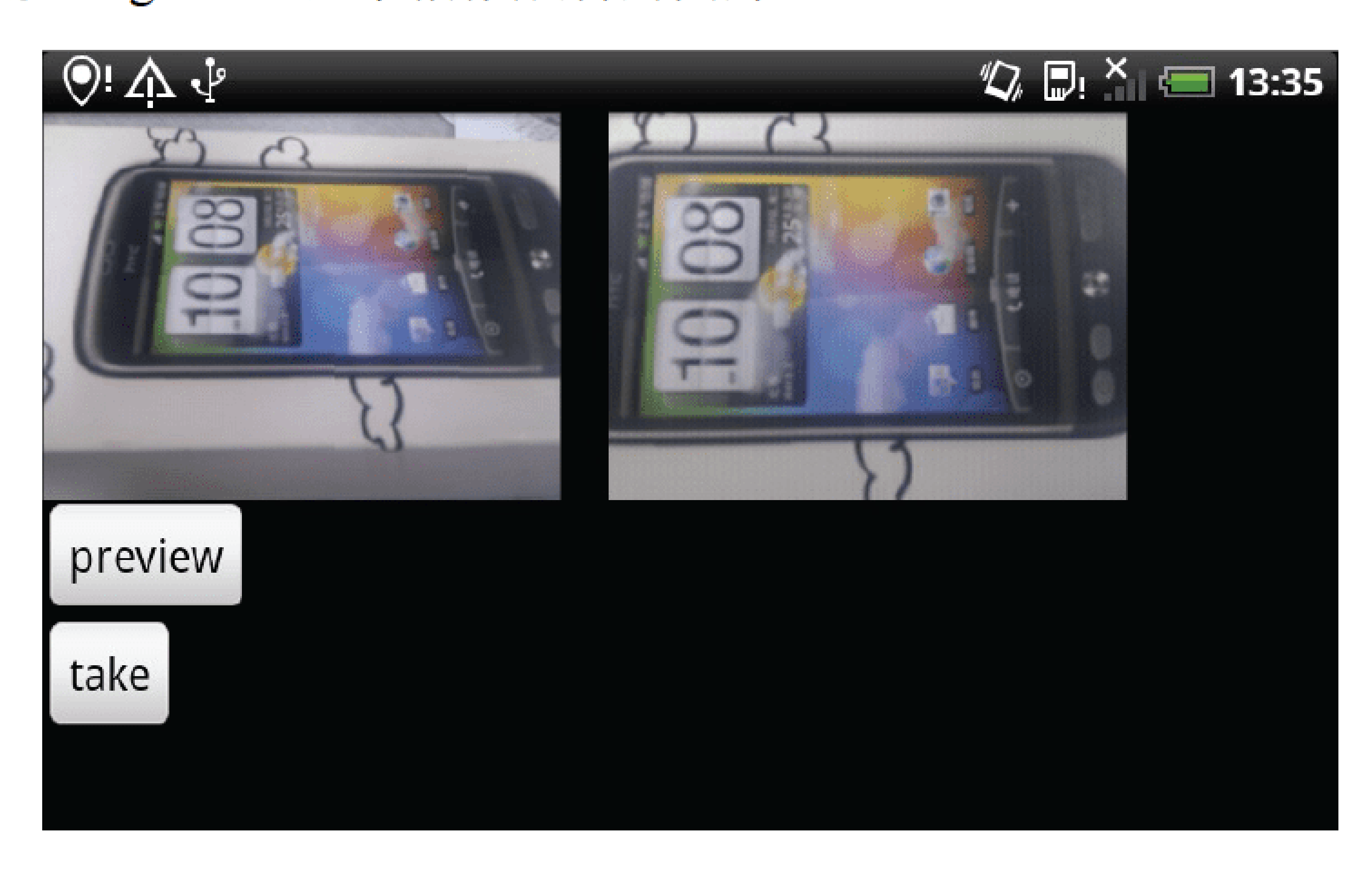


图 10.3　拍摄效果

当然，如果读者朋友们有兴趣可以进一步增强该应用的功能，如添加自动调焦的功能，可以添加如下代码以达到目的：

```
private AutoFocusCallback afc = new Camera.AutoFocusCallback()
    {
        @Override
        public void onAutoFocus(boolean focused, Camera arg1)
        {
            if (focused)
            {
                takePicture();
            }
        }
    };
```

注意粗体部分的代码，该方法为自动调焦，第一个参数为 true 时调焦完毕，这时我们就开启拍摄功能，得到的拍摄效果会更好一些。

好了，拍摄功能我们就讲解到这里，如果读者朋友们有兴趣可以继续钻研，无论是 Camera 的使用，亦或是 SurfaceView 的使用都有很大的探索空间。下一节我们将讲解通过摄像头录制视频。

10.4　学习视频处理

摄像头除了可以用作照相之外，我们还可以利用它拍摄视频。使用 Android SDK 中提

供的 MediaRecorder 可以很方便地操作摄像头进行视频的录制。该类我们在之前的音频处理中已经接触过，学习了使用它录制音频。实际上 MediaRecorder 可以同时进行音频和视频的录制。

10.4.1　学习录制视频

录制视频与照相一样，同样需要分为两个步骤，第一步依然是渲染一个 SurfaceView 用以展示预览，第二步就是通过 MediaRecorder 录制视频文件了。怎样定义 SurfaceView 我们在上一节已经有过讲解，不同的是在绑定时我们要使用方法：

```
MediaRecorder.setPreviewDisplay(Surface sv)
```

与 Camera.setPreviewDisplay()方法类似，不同的是这里我们需要传递的参数不再是 SurfaceHolder，而是 Surface。那么怎么得到这个 Surface 呢？很简单，它就藏在 SurfaceHolder 里，使用 SurfaceHolder.getSurface()就可以得到 Surface 对象了，这两个方法的效果是类似的。

预览部分就讲解到这里了，录制部分需要以下 11 个步骤：

（1）获得 MediaRecorder 对象。
（2）设置录制设备。
（3）设置输出格式。
（4）设置录制大小（可选）。
（5）设置录制时帧率（可选）。
（6）设置压缩格式。
（7）设置输出文件。
（8）准备录制。
（9）开始录制。
（10）停止录制。
（11）释放资源。

接下来我们详细讲解各个步骤的功能以及实现：

（1）获得 MediaRecorder 对象：

```
recorder = new MediaRecorder();
```

（2）设置录制设备：

```
MediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
```

（3）设置输出格式：

```
MediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
```

这里的参数还可以设置为 MPEG_4、DEAFULT、RAW_AMR，不过在 Android SDK 中官方强烈建议音频编码为 AMR、视频编码为 H263 时使用 THREE_GPP 为输出格式。

（4）设置录制大小：

```
MediaRecorder.setVideoSize(800, 480);
```

录制大小的设置是一个可选属性，你可以选择不进行设置，每个手机的默认大小都不同。这里的 VideoSize 设置关系到录出来的视频清晰与否，如果设置为 1024×720 也就是我们平常所说的高清了。更久以前一般手机的录制大小都是 176×144 或者 352×288 等。

（5）设置录制时帧率：

```
MediaRecorder.setVideoFrameRate(25);
```

这里的帧率设置同样是一个可选属性，根据 Android SDK 的描述，部分设备设置该属性无效，因为系统的摄像头是自动帧率，这个时候该方法则设置了最高帧率。根据笔者的实践，发现 HTC Desire 设置帧率的确是无效的，它是一个在 20 左右浮动的值，但是笔者同样发现该方法并没有限制录制时的最高帧率。当然这个属性在录制时设置与否和最后的效果并没有太大的关系，我们只能期待以后的 SDK 是否有所提高。

（6）设置编码格式：

```
MediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.H264);
```

这里的参数还可以设置为 MediaRecorder.VideoEncoder.H263 或者 MediaRecorder.VideoEncoder.MPEG_4_SP。

（7）设置输出文件：

```
MediaRecorder.setOutputFile(path);
```

这里的参数是文件的有效路径。

（8）准备录制：

```
MediaRecorder.prepare()
```

MediaRecorder 必须要先准备才可以开始录制，否则会出现非法状态异常，因为在准备时，Java 层会通过 JNI 调用进行一些摄像头的初始化。

（9）开始录制：

```
MediaRecorder.start()
```

（10）停止录制：

```
MediaRecorder.stop()
```

（11）释放资源：

```
MediaRecorder.release()
```

到这里一个完整的流程就结束了。开发人员需要注意的是，在使用结束后不要忘记释放资源，否则会造成程序运行缓慢，严重时甚至会出现死机。

10.4.2　实例——录制视频

通过上小节的学习我们已经基本掌握了视频录制的相关方法，接下来依然通过一个实例将学习到的知识进行一个巩固和整合。本实例设计在屏幕上有一个预览框，单击“录制”按钮后开始录制，录制的文件保存为 3GP 格式。

接下来进行界面设计，界面很简单，只需要一个 SurfaceView 和两个按钮，一个用来

单击开始预览，一个单击开始录制，xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <SurfaceView
    android:id="@+id/surfaceView1"
    android:layout_width="480px"
    android:layout_height="320px"
    />
    <Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="record"
    android:textSize="18sp"
    />
      <Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="stop"
    android:textSize="18sp"
    />

</LinearLayout>
```

接下来进行 Java 部分的代码编写，整体设计：

```
package com.wes.demo;

import ...                  //省略部分导入
import android.media.MediaRecorder;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class MediaDemo extends Activity implements SurfaceHolder.Callback
{
    SurfaceView sv;
    SurfaceHolder holder;
    Button takeBtn;
    Button stopBtn;
    String path = "/sdcard/test.3gp";
    MediaRecorder  recorder;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.main);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_
        LANDSCAPE);
        initView();

         takeBtn.setOnClickListener(new OnClickListener()
         {

             @Override
             public void onClick(View arg0)
```

```
            {
                initRecorder();                    //初始化 recorder
                record();                          //开始录制
            }
        });

    stopBtn.setOnClickListener(new OnClickListener()
        {

            @Override
            public void onClick(View arg0)
            {
                recorder.stop();                   //停止录制
                recorder.release();                //释放资源
            }
        });
}
}
```

当 record 按钮被单击时开始执行初始化 Recorder 和录制工作，当 stop 按钮被单击时停止录制并释放资源。initView()方法中完成组件的实例化和 SurfaceView 以及 SurfaceHolder 的设置与 10.3 节类似，这里不再给出相关代码，接下来分析 initRecorder()函数。

（1）初始化 Recorder：

```
private void initRecorder()
    {
    recorder = new MediaRecorder();                         //获得 recorder 对象
    recorder.setPreviewDisplay(holder.getSurface());   //设置预览框
        recorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
                                                        //设置录制源
        recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
//设置输出格式
//      recorder.setVideoFrameRate(25);                 //设置帧率，部分设备无效
        recorder.setVideoSize(800, 480);
                                    //设置视频大小，必须在编码格式之前，否则无效
        recorder.setVideoEncoder(MediaRecorder.VideoEncoder.H264);
                                                        //设置编码格式
        recorder.setOutputFile(path);                   //设置输出文件
    }
```

读者朋友们需要注意的是这里的设置顺序，录制时一定要先设置录制源，其次设置输出格式，接着如果要设置大小必须在编码格式之前，否则设置会无效。初始化完毕我们就可以开始录制了。

（2）录制视频：

```
public void record()
    {
    try
        {
            recorder.prepare();
            recorder.start();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
```

再次强调，必须要准备之后才可以执行 start()方法，否则会出现异常，prepare()方法也必须在前面的设置完毕后才可以调用，否则也会出现异常，程序运行后效果如图 10.4 所示。

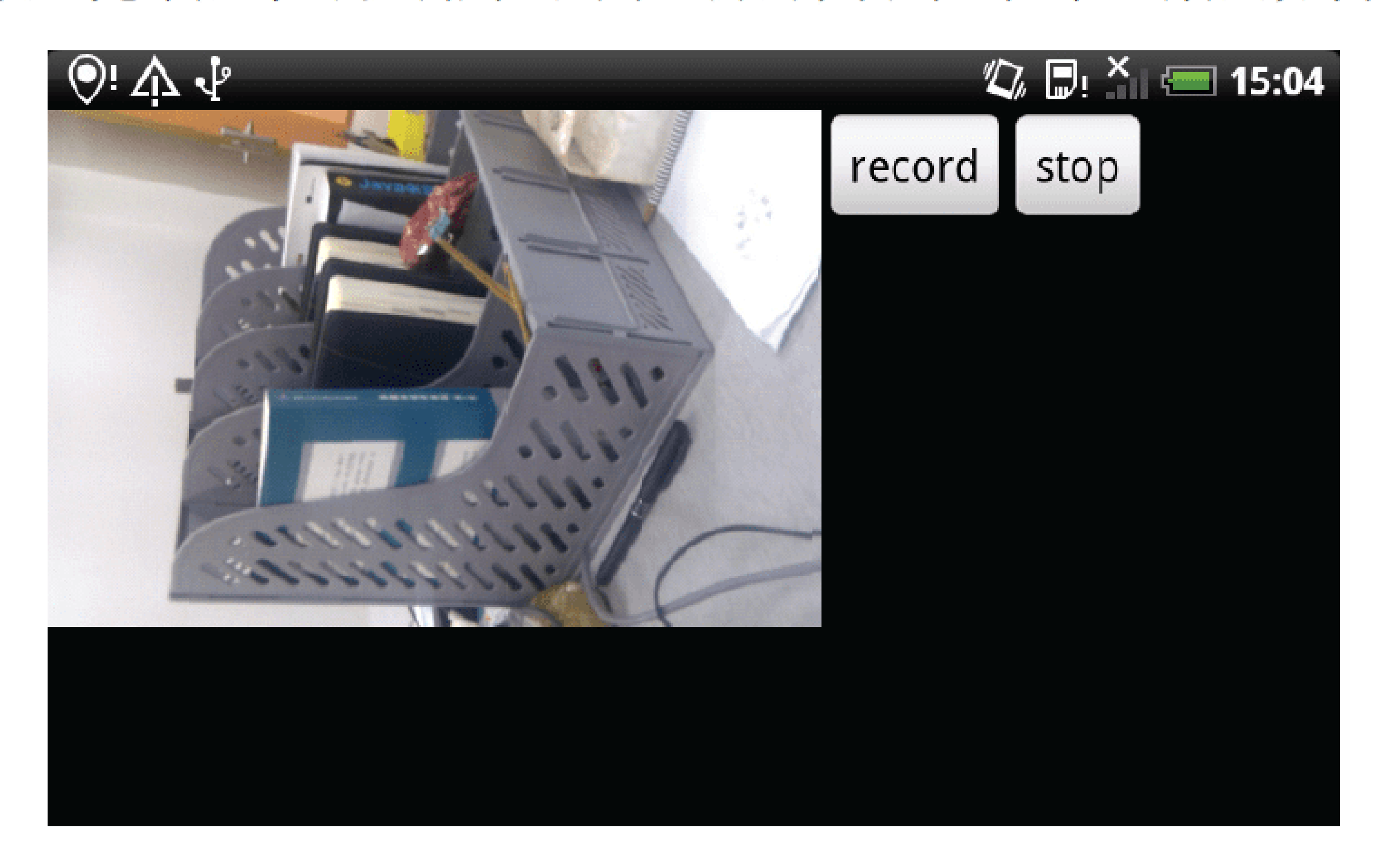


图 10.4　录制时界面

10.4.3　学习播放视频

上一小节我们完成了视频的录制，那么这一小节要学习的内容就是视频的播放。播放视频时，我们使用与 MediaRecorder 类相对应的 MediaPlayer 类。显示的组件我们可以使用 VideoView 来完成，但是使用该组件显得太普通、太统一，我们还是希望能够自己定制界面。

所以本小节中我们依然用 SurfaceView 完成视频的显示。SurfaceView 的使用这里不再细说，接下来学习使用 MediaPlayer 进行视频播放的相关步骤：

（1）获得 MediaPlayer 对象。

（2）绑定播放组件。

（3）设置数据源。

（4）准备播放。

（5）开始播放。

（6）暂停播放。

（7）停止播放。

（8）释放资源。

这里的 8 个步骤显示了一个完整的使用 MediaPlayer 播放视频的流程。接下来详细讲解各个步骤的相关功能及语法：

（1）获得 MediaPlayer 对象，方法为：

```
player = new MediaPlayer();
```

（2）绑定播放组件：

```
MediaPlayer.setDisplay(SurfaceHolder sh);
```

这里的参数依然是 SurfaceHolder，与 setPreviewDispaly()方法相同。

（3）设置数据源：

```
MediaPlayer.setDataSource(String arg0);
```

这里已经是设置的最后一步了，把文件名作为参数传递给 MediaPlayer，接下来的工作就交给它吧，我们不再需要其他的设置了，接下来要做的就是控制 MediaPlayer 的状态了。

（4）准备播放：

```
MediaPlayer.prepare();
```

（5）开始播放：

```
MediaPlayer.start()
```

（6）暂停播放：

```
MediaPlayer.pause()
```

（7）停止播放：

```
MediaPlayer.stop();
```

（8）释放资源：

```
MediaPlayer.release();
```

这里的状态与 MediaRecorder 类似，不同的是多出了一个暂停状态，在暂停状态下重新开始播放则执行 start()；不再需要 prepare()，如果又执行了 prepare()则出现状态异常。只有在释放资源以后程序才需要 prepare()。

10.4.4　实例——自制视频播放器

本小节就通过 MediaPlayer 结合 SurfaceView 制作一个自定义的播放器。播放的资源就是上一小节中录制的 3GP 文件，界面设计是一个 SurfaceView 和 3 个按钮；分别用来开始播放、暂停播放和停止播放。

本实例的重要方法为：

```
MediaPlayer.setDisplay(SurfaceHolder sh);
```

接下来就观察界面设计，其中 SurfaceView 和一组按钮在一个线性布局中垂直布局，3 个按钮在一个线性布局中水平布局。xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
     >
    <SurfaceView
    android:id="@+id/surfaceView1"
    android:layout width="480px"
    android:layout height="680px"
    />
    <LinearLayout
    android:orientation="horizontal"
      >
        <Button
```

```
        android:id="@+id/button1"
         android:text="play"
             />
         <Button
        android:id="@+id/button3"
         android:text="pause"
             />
         <Button
        android:id="@+id/button2"
       android:text="stop"
             />
    </LinearLayout>
</LinearLayout>
```

代码中，这里只列出了一些比较重要的属性，其他属性，如宽度和高度等由于篇幅原因就不再列出。接下来分析 Java 部分的代码。Java 代码的整体设计：

```
package com.wes.demo;

import ...                                       //省略部分导入
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class MediaDemo extends Activity implements SurfaceHolder.Callback
{
    SurfaceView sv;
    SurfaceHolder holder;
    Button playBtn;
    Button stopBtn;
    Button pauseBtn;
    String path = "/sdcard/test.3gp";            //文件路径
    MediaPlayer  player;
    boolean isPrepared = false;                  //是否准备完毕标签
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.main);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_
        PORTRAIT);
        initView();
         initPlayer();                           //初始化 player

         playBtn.setOnClickListener(new OnClickListener()
         {
                        @Override
             public void onClick(View arg0)
             {
                 play();                         //开始播放
             }
         });

         pauseBtn.setOnClickListener(new OnClickListener()
         {
                        @Override
             public void onClick(View arg0)
             {
                 player.pause();                 //暂停播放
```

```
            }
        });

    stopBtn.setOnClickListener(new OnClickListener()
        {
                        @Override
            public void onClick(View arg0)
            {
                if (player.isPlaying())
                {
                    player.stop();              //停止播放
                    isPrepared = false;
                }
player.release();                               //释放资源
            }
        });
}
]
```

在整体设计部分，我们需要注意的是各个状态之间的关系，不合法的状态切换会造成异常。“停止”按钮被单击时先判断是否正在播放，如果是才进行停止和释放资源。initView()函数由读者自己去完成，这里不再给出代码，通过之前的几个程序，相信读者现在对SurfaceView 的使用没有什么问题了。我们接下来重点讲解 initPlayer()。

（1）初始化播放器：

```
private void initPlayer()
{
    player = new MediaPlayer();                         //获得播放器对象
    player.setDisplay(holder);                          //绑定显示组件
    try
    {
        player.setDataSource(path);                     //设置数据源
    }
    catch (IllegalArgumentException e)
    {
        e.printStackTrace();                            //非法参数异常
    }
    catch (IllegalStateException e)
    {
        e.printStackTrace();                            //非法状态异常
    }
    catch (IOException e)
    {
        e.printStackTrace();                            //I/O 异常
    }
}
```

使用 MediaPlayer 的好处就是可以避免繁琐的参数设置，只要如上的 3 个步骤就可以完成 Player 的初始化了，以上的 3 个步骤就是本实例的核心代码。接下来完成播放功能。

（2）播放功能：

```
public void play()
{
    try
        {
        if (!isPrepared)
        {
```

```
                player.prepare();               //如果没有准备好则准备播放器
                isPrepared = true;              //将已准备好标签设置为 true
            }
            player.start();                     //开始播放
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
}
```

为了防止暂停后在单击“播放”按钮重新准备导致异常，这里设置了一个条件判断，如果准备好则不再进行 prepare()方法。接下来就赶快运行程序，效果如图 10.5 所示。

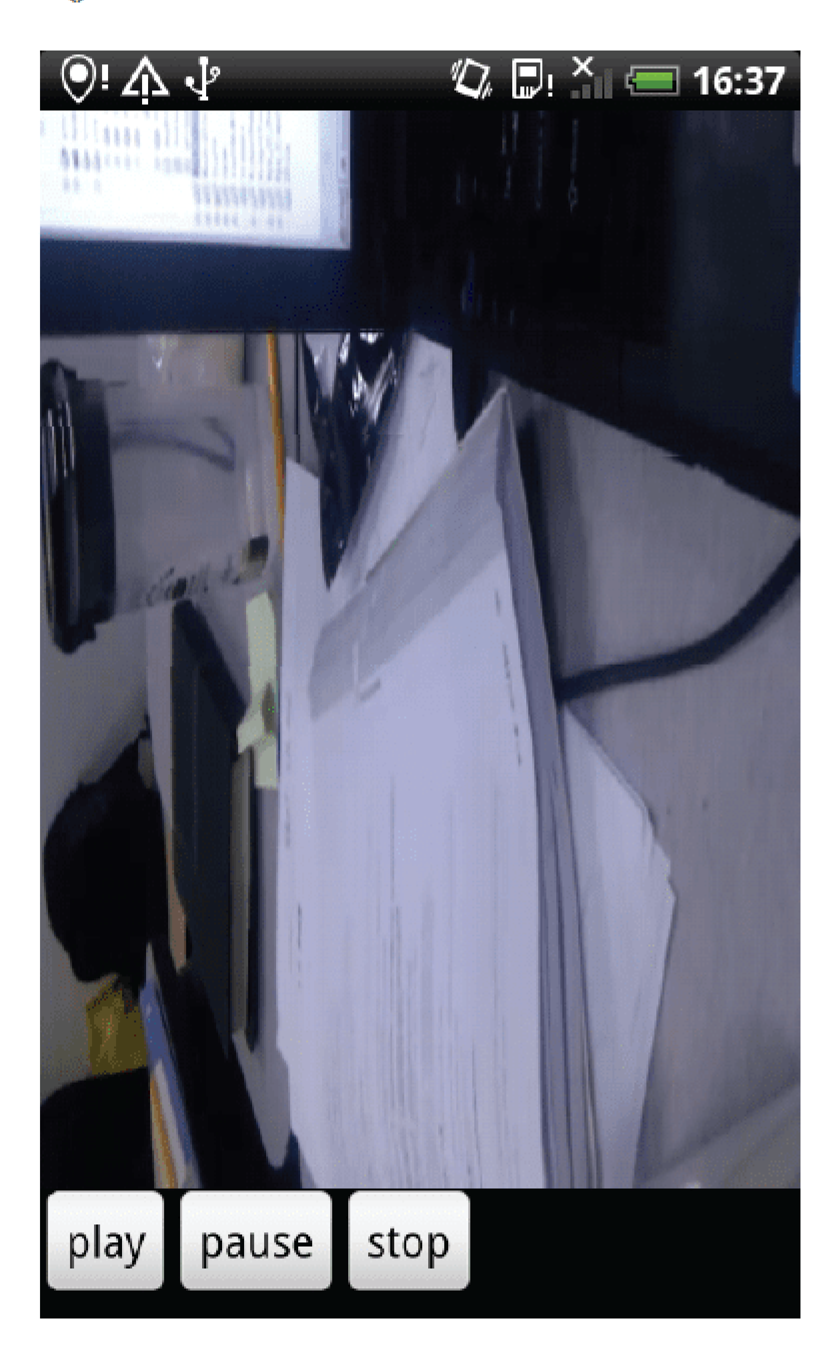


图 10.5　自制播放器

本例只是实现了播放器的最基本功能，如果读者有兴趣可以进一步开发，比如增加进度条，显示播放到哪里了。甚至可以增加一个拖动条组件，拖动视频到希望的位置，可以使用 MeidaPlayer.seekTo()方法实现该功能。

10.5　小　　结

本章讲解了利用麦克风录制音频、利用摄像头拍照和录制视频。学习了 MediaRecorder 和 MediaPlayer 的使用，更学习了 AudioRecord 和 AudioTrack 类进行音频裸数据的处理。本章重点是各个类的参数设置和各个参数的意义，难点是播放器和录制器的状态控制。

在本章开头就提到多媒体是一个大课题，还有更多的内容需要读者自己去挖掘和学习，本章仅仅介绍了一些基础知识。下一章我们将学习 Android 网上冲浪相关的知识。

第 11 章　Android 网上冲浪

Android 手机除了最基本的电话和短信功能之外，我想大家用的最多的就是上网功能了。Android 作为一个网络操作系统，对网络的支持无疑是非常强大的，本章就将学习网络方面的 API。通过本章的学习，读者朋友们将能够编写自己的 HTTP 客户端，学会使用 HTTP 协议发送和接收数据，同时将会编写自己的个性浏览器。

11.1　使用 HttpUrlConnection

在网络中最常见的数据传输方式就是超文本传输协议——HTTP（Hyper Text Transfer Protocol）了。本节我们将使用 HttpUrlConnection 类进行数据的接收和发送。在传输时有两种方法可以选择，分别是 GET 和 POST。这两种方法在使用时是有区别的，我们要根据选择的方法进行相应的代码编写。

11.1.1　使用 GET 方法

在 HTTP 协议中 GET 常被用来查询数据，它的参数可以直接写在 URL 中，如：http://192.168.0.1: 8080/index.jsp?id=123456。

以上的 URL 中就包含了参数 id=123456 的信息，这样我们在查询时将非常方便。GET 方法也是 HttpUrlConnection 的默认连接方法。接下来我们就开始学习怎样使用 HttpUrl-Connection 进行网络数据的传输。

使用 HttpUrlConnection 的 GET 方法需要如下 6 个步骤：

（1）新建 URL。

（2）得到 HttpUrlConnection 连接对象。

（3）设置该连接对象。

（4）得到输入流。

（5）从流中读取返回的结果，进行处理。

（6）关闭流。

那么这 6 个步骤又是怎样实现的呢？让我们也一步一步往下看。

1．新建URL

URL 的全称是资源描述符，它的作用是描述一个网上的资源。看上去似乎比较深奥，但是实际上，它就是我们平常的网址，如本章开始的举例：http://192.168.0.1:8080/index.jsp?id=123456。

以上字符串作为参数传递到 URL 的构造函数中就可以新建一个 URL 对象了，代码如下：

```
URL url = new URL("http://192.168.0.1:8080/index.jsp?id=123456");
```

2．得到连接对象

我们不使用 New 方法得到连接对象，而是通过第一步新建的 URL 对象的 openConnection()方法得到，代码如下：

```
HttpURLConnection connection = (HttpURLConnection)url.openConnection();
```

3．设置连接对象

我们可以设置取得的连接对象的一系列属性，最常用的包括：

（1）允许读取：

```
URLConnection.setDoInput(boolean newValue)
```

（2）允许写入：

```
URLConnection.setDoOutput(boolean newValue)
```

（3）设置请求方法：

```
HttpURLConnection.setRequestMethod("GET") throws ProtocolException
```

（4）设置超时时间：

```
URLConnection.setConnectTimeout(int timeout)
```

（5）设置是否允许使用缓存：

```
URLConnection.setUseCaches(boolean newValue)
```

4．得到输入流

从连接中我们可以得到输入流，其方法为：

```
URLConnection.getInputStream() throws IOException
```

一般情况下，我们得到输入流还需对其进行包装，这样可以使 IO 操作更具效率。

5．从流中读取结果

这一步想必不要多说了，不同的流方法不一样。笔者比较推荐 BufferedReader 的 readLine()方法，使用方便且效率不错。

6．关闭流

使用完流后一定要养成关闭的好习惯，流就像是自来水，大家肯定有使用完自来水笼头后随手关闭的习惯吧！并且工作也非常简单，调用 close()方法就可以了。

11.1.2　使用 POST 方法

POST 方法与 GET 方法不同，它的参数不能直接写在 URL 中，而是在 HTTP 的包体

中，具体实现就是要通过 OutputStream 写数据。除此之外，POST 与 GET 方法大同小异，使用步骤如下：

（1）新建 URL 对象。

（2）获得 HttpUrlConnection 连接对象。

（3）设置连接对象，注意设置请求方法为 POST。

（4）获得输出流，写入数据。

（5）获得输入流，读取返回的数据。

（6）关闭流。

将以上步骤与 GET 方法相比较，不难发现只有第 3 步以及第 4 步存在差别，在第 3 步中注意使用方法：

```
HttpURLConnection.setRequestMethod(“POST”)
```

在第 4 步中获得输出流，方法为：

```
URLConnection.getOutputStream() throws IOException
```

写入数据时要注意对数据进行编码，方法为：

```
URLEncoder.encode(String s, String enc)  throws  UnsupportedEncodingException
```

这里的两个参数第一个是需要传输的内容，第二个是编码方式。

该方法返回的值是一个 String 类型字符串，这就是我们可以在网上传输的数据了。

最后不要忘记使用完毕后调用 close()方法关闭流。

11.1.3　通过实例学习 HttpUrlConnection

本小节将通过一个实例来帮助读者进一步掌握 HttpUrlConnection 的使用。首先创建一个工程，在 Activity 的布局文件中添加如表 11-1 所示的 3 个组件。

表 11-1　布局包含的组件

类　型	ID	意　　义
TextView	Tv	显示返回的数据，最好包裹一层 ScrollView，因为数据一般比较长
Button	Get	通过 GET 方法使用 HttpUrlConnection
Button	Post	通过 POST 方法使用 HttpUrlConnection

接着进行 Java 部分的代码编写。

1．整体设计

Java 整体设计的框架如下，它包括 doGet()和 doPost()方法的实现：

```
public class HttpDemo extends Activity {
    TextView tv;
    Button btn1;
    Button btn2;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.tv);
        btn1 = (Button) findViewById(R.id.post);
        btn2 = (Button) findViewById(R.id.get);

        OnClickListener listener = new OnClickListener()
         {
             @Override
             public void onClick(View v)
             {
                 int id = v.getId();
                 if (id == R.id.post)
                 {
                     doPost();
                 }
                 else
                 {
                     doGet();
                 }
             }
         };
         btn1.setOnClickListener(listener);
         btn2.setOnClickListener(listener);
}
}
```

2．实现doGet()方法

该方法用来通过 GET 方法进行 HTTP 请求，在进行网络编程时，最好将代码写在try{…}catch(){…}语句块中，因为处理网络编程时随时随地会遇到异常状况。例如，突然网络断开了、服务器维护关闭了等等。

```
public void doGet()
    {
    try
        {
            //新建 URL
            URL url = new URL("http://www.baidu.com");
            //创建 URL 连接
            HttpURLConnection connection = (HttpURLConnection)url.
            openConnection();
            //设置该连接允许读取
            connection.setDoInput(true);
            //设置该连接允许写入
            connection.setDoOutput(true);
            //设置超时
            connection.setConnectTimeout(1000);

            //得到连接的输入流
            InputStreamReader isr = new InputStreamReader(connection.
            getInputStream());
            //再次包装为缓冲流
            BufferedReader br = new BufferedReader(isr);
            //用来存放临时读取的行
            String tempResult = null;
            //用来保存全部的结果
            String result = null;
            //不停地读取行，直到结束
```

```
            while((tempResult = br.readLine()) != null)
            {
                //将结果保存
                result += tempResult + "\n" ;
            }
            //显示结果
            tv.setText(result);
            br.close();
            isr.close();
        }
        catch (IOException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

3. 实现doPost()方法

该方法实现了以 POST 进行 HTTP 请求，POST 方法不能使用缓存，同时使用该方法必须使用 OutputStream 进行数据写操作。代码如下：

```
public void doPost()
{
    try
    {
        //新建 URL
        URL url = new URL("http://5billion.com.cn/post.php ");
        //创建 URL 连接
        HttpURLConnection connection = (HttpURLConnection)url.
        openConnection();
        //设置该连接允许读取
        connection.setDoInput(true);
        //设置该连接允许写入
        connection.setDoOutput(true);
        //设置连接方法为 POST
        connection.setRequestMethod("POST");
        //设置不能使用缓存（POST 方法不可以使用缓存）
        connection.setUseCaches(false);
        //开始连接，在连接前请确认设置工作全部完毕
        connection.connect();
        //得到输出流
        DataOutputStream dos = new DataOutputStream(connection.
        getOutputStream());
        //需要写的参数
        String params = URLEncoder.encode("name=123456", "gb2312");
        //将参数写入到流中
        dos.write(params.getBytes());
        //将流中的数据全部写入
        dos.flush();
        //关闭流
        dos.close();
        //得到输入流
        InputStreamReader isr = new InputStreamReader(connection.
        getInputStream());
        //包装为缓冲流
        BufferedReader br = new BufferedReader(isr);
```

```
        //用来存放临时读取的行
        String tempResult = null;
        //用来保存全部的结果
        String result = null;
        //不停地读取行，直到结束
        while((tempResult = br.readLine()) != null)
        {
            //将结果保存
            result += tempResult + "\n" ;
        }
        //显示结果
        tv.setText(result);

    }
    catch (MalformedURLException e)
    {
        //如果 URL 不正确则抛出该异常
        e.printStackTrace();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
```

以上的代码都已经进行了相关的注释，相信大家理解起来应该没有什么问题，其中：

```
BufferedReader.readLine()
```

该方法的返回值是每一行的数据，如果没有数据了则返回空，所以我们可以利用这点特性作为循环的条件，将所有的数据都读取出来。

最后不要忘记在注册文件中添加权限许可，修改后的注册文件如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.wes.httpdemo"
     android:versionCode="1"
     android:versionName="1.0">
   <uses-sdk android:minSdkVersion="8" />

   <application android:icon="@drawable/icon" android:label="@string/
   app name">
      <activity android:name=".HttpDemo"
             android:label="@string/app name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

   </application>
   <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

注意其中的粗体部分，注意其添加的位置需要在 application 节点之外。

运行以上程序，效果如图 11.1 所示。




单击 POST 按钮后



单击 GET 按钮后

图 11.1　运行效果图

11.2　使用 HttpClient

为了避免繁琐的 HttpUrlConnection 的设置，Android SDK 为我们提供了 Apache 的 HttpClient 以简化操作，它将 GET 方法的连接封装成了 HttpGet 类，将 POST 方法的连接封装成了 HttpPost 类。这样我们使用起来就更方便了。

11.2.1　使用 HttpClient 进行 GET 连接

使用 HttpClient 进行 GET 方法的连接操作非常简单，按照如下步骤编写就可以了：

（1）新建 URI。

（2）新建 GET 型请求。

（3）新建 Http 客户端。

（4）使用客户端执行请求。

（5）处理返回的结果。

接下来我们进行每个步骤的详细解析。

1．新建URI

这里的 URI 并不是 URI 类，而是 String 类型的 uri 字符串就可以了，如：

```
String uri = "http://5billion.com.cn/post.php?name=abcd";
```

2．新建GET型请求

新建该请求时，只需使用 HttpGet 类就可以了，使用构造方法为：

```
HttpGet.HttpGet(String uri)
```

这里的参数就是第一步中的 String 类型的 uri。

3．新建Http客户端

新建客户端时，一般情况下使用默认的客户端就可以了，我们可以使用无参数的构造函数得到其操作对象：

```
DefaultHttpClient.DefaultHttpClient()
```

4．使用客户端执行请求

经过以上 3 个步骤我们得到了客户端，并且也已经新建好了请求，可谓是万事俱备，接下来的工作就是执行了，方法为：

```
HttpClient.execute(HttpUriRequest request) throws IOException,
ClientProtocolException
```

该方法的返回值是 HttpResponse 型，顾名思义，就是响应啦。

5．处理响应

得到了 HttpResponse 类型的响应后，在网络上的工作就完成了。接下来的事情就是在本地进行响应的处理了，如：

```
HttpResponse.getStatusLine()            //得到状态行
HttpResponse.getEntity()                //得到结果
```

11.2.2　使用 HttpClient 进行 POST 连接

同样地，使用 POST 连接与 GET 连接不同之处就是参数的传递，在 POST 中传递参数需要使用 NameValuePair 类，NameValuePair 翻译为中文就是“名值对”了，这想必大家肯定不陌生了，其实质也就是 HashMap 罢了。

使用 POST 型连接与 GET 型连接大体类似，唯一的不同就是请求的建立，步骤如下：

1．新建HttpPost类型的请求

同样地，我们使用带有 Uri 参数的构造方法创建 HttpPost 请求：

```
HttpPost.HttpPost(String uri)
```

2．新建保存参数的数据结构

刚才我们已经提到，在 POST 中保存参数使用的是 NameValuePair 类，仅仅保存一个名值对肯定是不够的，所以真正传递的是一个列表，列表中的每一个对象都是名值对，举例如下：

（1）新建一个列表用来保存名值对：

```
List<NameValuePair> params = new ArrayList<NameValuePair>();
```

（2）接着新建真正的数据，以名值对的形式保存，方法为：

```
BasicNameValuePair.BasicNameValuePair(String name, String value)
```

（3）将名值对添加到列表中：

```
List.add(NameValuePair object)
```

3．为参数设置编码方式

最后将之前新建好的 List 型参数添加到 HttpPost 对象中，使用方法为：

```
UrlEncodedFormEntity.UrlEncodedFormEntity(List<? extends NameValuePair>
parameters, String encoding) throws UnsupportedEncodingException
```

这里有两个参数，第一个参数无疑就是需要编码的参数，而第二个参数就是设置的编码方式了，可以使用 HTTP.UTF_8、HTTP.UTF_16 等。

4．将参数添加到请求中

使用 setEntity()方法可以将参数添加到请求中，语法格式如下：

```
HttpEntityEnclosingRequestBase.setEntity(HttpEntity entity)
```

接下来的工作就与之前一样了，我们再回顾一遍。

5．新建Http客户端

```
DefaultHttpClient.DefaultHttpClient()
```

6．使用客户端执行请求

```
HttpClient.execute(HttpUriRequest request) throws IOException,
ClientProtocolException
```

7．处理响应

这里的处理就需要按照应用的需求进行代码编写了，比如将结果以 String 形式显示，代码如下：

```
String result = EntityUtils.toString(httpResponse.getEntity());
tv.setText(result);
```

11.2.3　通过实例学习 HttpClient

接下来，依旧通过一个实例巩固 HttpClient 的使用方法。首先，新建一个工程，在工程中创建一个 Activity，在 Activity 的布局文件中添加如表 11-2 所示的 3 个组件。

接着在 Java 代码中进行功能代码的编写。

1．整体设计

Java 整体设计的框架如下：

表 11-2　HttpClient布局包含的组件

类　型	ID	意　　义
TextView	Tv	显示返回的数据
Button	Get	通过 GET 方法使用 HttpClient
Button	Post	通过 POST 方法使用 HttpClient

```
public class HttpDemo extends Activity {
    TextView tv;
    Button btn1;
    Button btn2;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      tv = (TextView) findViewById(R.id.tv);
      btn1 = (Button) findViewById(R.id.post);
      btn2 = (Button) findViewById(R.id.get);

      OnClickListener listener = new OnClickListener()
       {
           @Override
           public void onClick(View v)
           {
               int id = v.getId();
               if (id == R.id.post)
               {
                   doPost();
               }
               else
               {
                   doGet();
               }
           }
       };
       btn1.setOnClickListener(listener);
       btn2.setOnClickListener(listener);
}
}
```

接下来的任务就是分别完成 doPost()方法和 doGet()方法了，首先我们完成 doGet()部分。

2. 实现doGet()

按照 11.2.1 小节中讲解的 5 个步骤，我们可以很方便地编写出如下代码：

```
public void doGet()
    {
    //新建 URL
    String url = "http://5billion.com.cn/post.php?name=abcd";
    //新建 GET 型的请求
    HttpGet httpRequest = new HttpGet(url);
    //新建 HTTP 客户端
    HttpClient httpClient = new DefaultHttpClient();
    try
        {
        //执行请求返回结果
            HttpResponse httpResponse = httpClient.execute(httpRequest);
```

```
            //判断结果状态
           if (httpResponse.getStatusLine().getStatusCode() == HttpStatus.
            SC_OK)
            {
                //得到结果内容
                String result = EntityUtils.toString(httpResponse.
                getEntity());
                tv.setText(result);
            }
            else
            {
                tv.setText("应答错误:"+httpResponse.getStatusLine().
                toString());
            }
        }
        catch (ClientProtocolException e)
        {
            // 客户端协议异常
            e.printStackTrace();
        }
        catch (IOException e)
        {
            // IO 异常
            e.printStackTrace();
        }
    }
```

其中，得到相应的状态码时，HttpStatus.SC_OK 表示成功，它对应的值是 200，还有更多的状态码请参考 API 文档。

3．实现doPost()

使用 POST 时一定要注意参数的添加，HTTP 请求中的参数是需要设置编码方式的。代码如下所示：

```
public void doPost()
    {
    //新建 URL
    String url = "http://www.baidu.com";
    //新建 POST 类型的请求
    HttpPost httpRequest = new HttpPost(url);
    //新建需要传递参数的数据结构
    List<NameValuePair> params = new ArrayList<NameValuePair>();
    //新建键值对
    BasicNameValuePair pair1 = new BasicNameValuePair("param", "AaBbCcDdEe");
    //将数据添加到键值对中
    params.add(pair1);
    try
        {
        //设置编码方式
        HttpEntity entity = new UrlEncodedFormEntity(params, HTTP.UTF_8);
            httpRequest.setEntity(entity);
            //新建 HTTP 客户端
            HttpClient httpClient = new DefaultHttpClient();
            //执行请求得到响应
            HttpResponse httpResponse = httpClient.execute(httpRequest);
            //判断响应的状态是否成功
```

```
            if (httpResponse.getStatusLine().getStatusCode() == HttpStatus.
             SC_OK)
             {
                 //得到结果字符串
                 String result = EntityUtils.toString(httpResponse.
                 getEntity());
                 tv.setText(result);
             }
             else
             {
                 tv.setText("应答错误:"+httpResponse.getStatusLine().
                 toString());
             }
        }
        catch (UnsupportedEncodingException e)
        {
            //字符集编码不支持则捕获此异常
            e.printStackTrace();
        }
        catch (ClientProtocolException e)
        {
            //客户端协议异常
            e.printStackTrace();
        }
        catch (IOException e)
        {
            // IO 异常大家肯定不陌生了
            e.printStackTrace();
        }
    }
```

到这里代码就完成了。最后不要忘记添加 INTERNET 的权限许可：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

接下来运行代码，看看能否成功运行吧！运行后效果如图 11.2 所示。



单击 POST 按钮后　　　　单击 GET 按钮后

图 11.2　HttpClient 使用效果图

从图 11.2 我们可以得到结论：使用 POST 方法连接百度后得到的结果是不成功的。给出的提示是 HTTP/1.1 501 没有实现。

11.3　自制 Web 浏览器

相信只要有 Android 手机的读者必定对手机中的浏览器不陌生，那么这些功能强大的浏览器是怎样完成的呢？我们能不能编写出一个自定义的浏览器呢？答案是当然可以！通过使用 Android SDK 为我们提供的 WebView 就可以实现这个目的。本节将详细讲解 WebView 的使用，并带领读者完成自己的 Web 浏览器。

11.3.1　使用 WebView

WebView 相当于手机一个嵌入式的浏览器，可以加载并显示网页，它使用了 WebKit 渲染引擎。使用 WebView 主要步骤为：

（1）在布局文件中添加 WebView 组件。

（2）在 Activity 中实例化该组件。

（3）设置 WebView 客户端，如果不设置则使用内置的浏览器。

（4）加载 Url，显示网页。

通过以上步骤我们发现，其实使用 WebView 并没有想象中那么难。只要按照以上 4 步进行代码的编写，我们很快就能拥有自己的浏览器了。接下来我们就开始详细讲解每个步骤的具体方法。

1．在布局中添加WebView组件

添加 WebView 组件时，我们需要使用<WebView>节点，然后在其属性中添加必要的宽度、高度、id 等信息就可以了，一个最简单的 WebView 组件的 xml 代码如下：

```
<WebView
        android:layout width="fill parent"
    android:layout height="wrap content"
        android:id="@+id/wv1"
/>
```

2．在代码中实例化该组件

与其他所有的 View 对象一样，我们通过：

```
Activity.findViewById(int id)
```

得到其操作对象。

3．设置WebView客户端

得到 WebView 之后我们还需要为它设置客户端，如果没有设置自己的客户端，则默认使用 Android 自带的浏览器打开网页。既然我们要自定义 WebView 肯定要自己编写客户

端啦！方法如下：

```
WebViewClient client = new WebViewClient() {…};
```

在新建时，我们可以重写 WebViewClient 的一系列方法来达到自己的目的，表 11-3 列出了一些常用的方法。

表 11-3　WebViewClient常用方法

方　法　名	作　　用
onLoadResource	载入资源时回调该方法
onPageStart	页面开始加载时回调该方法
onPageFinish	页面加载结束
onReceiveError	接收错误时回调

4．加载Url，显示网页

将准备工作完成后，我们就可以加载 Url 了，方法如下：

```
WebView.loadUrl(String url)
```

当然，不要忘记在注册文件中添加相关权限，需要添加的代码如下：

```
<uses-permission android:name="android.permission.INTERNET" />
```

11.3.2　通过实例学习 WebView

通过上一小节的学习，我们已经基本掌握了编写 WebView 需要的知识。接下来我们就将以上的知识贯穿起来，一起来编写一个自定义的 WebView 浏览器，并实现一些特别的功能，比如放大、缩小、前进、后退以及截屏等。

首先新建一个工程，在 Activity 的布局文件中添加表 11-4 中所示的组件。

表 11-4　浏览器中的组件

类　　型	ID	意　　义
EditText	Et	用来输入网址
Buton	Btn	单击后开始加载动作
WebView	Wv	用于显示网页

关于这些组件怎样布局，这个问题仁者见仁智者见智。笔者的结构如下：

1．xml代码

xml 代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
   <TableLayout
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
```

```
    >
     <TableRow
     android:orientation="horizontal"
     android:layout width="fill parent"
     android:layout height="wrap content">
             <EditText
            android:layout width="fill parent"
            android:layout height="wrap content"
            android:id="@+id/et1"
            android:text="http://www.google.com.hk"
            android:singleLine="true"
            />
            <Button
            android:layout_width="80dp"
            android:layout_height="wrap_content"
            android:id="@+id/btn1"
            android:textSize="20sp"
            android:text="go"
            />
        </TableRow>
    </TableLayout>
    <WebView
        android:layout width="fill parent"
        android:layout height="wrap content"
        android:id="@+id/wv1"
    />
</LinearLayout>
```

如果你愿意当然可以编写出更加个性的界面来。接下来让我们把主要的精力放在 Java 代码的功能实现上。

2. Java代码

在 Activity 中，我们首先要进行整体设计。

（1）整体设计

```
@Override
public void onCreate(Bundle savedInstanceState)
 {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       et = (EditText) findViewById(R.id.et1);
       btn = (Button) findViewById(R.id.btn1);
       //实例化组件
       wv = (WebView) findViewById(R.id.wv1);
       //得到 WebView 设置
       settings = wv.getSettings();
       //新建客户端
       WebViewClient client = new WebViewClient()
       {
       };
        //设置客户端，如果没有设置则调用内置的浏览器
       wv.setWebViewClient(client);
       //设置按钮监听事件
       btn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
```

```
                String url = et.getText().toString();
                if(URLUtil.isNetworkUrl(url))
                {
                //加载 Url
                    wv.loadUrl(url);
                }
                else
                {
                    Toast.makeText(getBaseContext(), "网址非法", Toast.
                    LENGTH_SHORT).show();
                }
            }
        });
}
}
```

事实上，以上代码就可以完成浏览器功能了，但是这样是不是显得简陋了一些呢？我们可以将浏览器制作得更加强大！

（2）完善 WebView 客户端

刚才我们提到，WebViewClient 有一系列的方法可以重写，以实现我们的目的，这里我们就重写 3 个方法，分别是：

- onPageStarted()
- onPageFinished()
- onLoadResource()

在页面开始时，提示用户正在载入；在加载资源时，显示正在加载的 Url；在页面加载结束时，将页面截屏并保存。实现的代码如下：

```
//重写页面开始加载方法
            @Override
            public void onPageStarted(WebView view, String url, Bitmap
            favicon)
            {
                Toast.makeText(view.getContext(), "正在载入...", Toast.
                LENGTH_SHORT).show();
                super.onPageStarted(view, url, favicon);
            }
            //重写页面加载结束方法
            @Override
            public void onPageFinished(WebView view, String url)
            {
                //得到网页的图片
                Picture pic = view.capturePicture();
                //得到图片的宽和高
                int width = pic.getWidth();
                int height = pic.getHeight();
                //如果宽或高为 0，则不再执行操作
                if (width * height == 0)
                    return;
                //新建位图用以存放数据
                Bitmap bitmap = Bitmap.createBitmap(width, height, Bitmap.
                Config.ARGB_8888);
                //新建画布，指定要画的位图对象
                Canvas canvas = new Canvas(bitmap);
                //将图片画到画布上
                pic.draw(canvas);
```

```
        try
        {
            //创建文件
            File file = new File("/sdcard/pic.jpg");
            if (file.exists())
                file.delete();
            file.createNewFile();
            //得到输出流
            FileOutputStream fos = new FileOutputStream(file);
            if (fos != null)
            {
                //将位图对象的数据压缩成 JPEG 格式的图片
                bitmap.compress(Bitmap.CompressFormat.JPEG, 90, fos);
            }
            Toast.makeText(view.getContext(), "照片保存成功! ",
            Toast.LENGTH SHORT).show();
        }
        catch (FileNotFoundException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        catch (IOException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        super.onPageFinished(view, url);
    }
    //重写加载资源方法
    @Override
    public void onLoadResource(WebView view, String url)
    {
        Toast.makeText(view.getContext(), "正在载入:"+url, Toast.
        LENGTH_SHORT).show();
        super.onLoadResource(view, url);
    }
```

此时，在你的 SD 卡中，应该已经保存有 pic.jpg 图片了。如果你还有更多的想法，当然可以重写更多的方法，或者在这些回调函数中实现其他的工作。此时运行程序我们可以得到效果如图 11.3 所示。





图 11.3　重写 WebViewClient

接着，我们继续实现网页的缩小、放大、前进、后退等常用功能。其实这些功能都只要调用一些简单的 API 就可以实现。

前进：

```
WebView.goForward()
```

后退：

```
WebView.goBack()
```

实现放大、缩小功能时，首先要得到 WebView 的设置对象，方法为：

```
WebView.getSettings()
```

接下来，缩小、放大都是通过调用同一个函数实现的：

```
WebSettings.setDefaultZoom(ZoomDensity zoom)
```

放大时，ZoomDensity 参数设置为：

```
WebSettings.ZoomDensity.CLOSE
```

缩小时，参数设置为：

```
WebSettings.ZoomDensity.FAR
```

默认时，参数为：

```
WebSettings.ZoomDensity.MEDIUM
```

为什么通过设置参数就可以实现放大、缩小的功能呢？实际上通过观察源代码，我们发现该参数实际上就是设置网页的 dpi，也就是每英寸的点数，点数密级了，那就相当于缩小了，反之亦然。部分源代码如下：

```
public enum ZoomDensity {
      FAR(150),        //240dpi
      MEDIUM(100),     //160dpi
      CLOSE(75);       //120dpi
      ZoomDensity(int size) {
         value = size;
      }
      int value;
   }
```

我们通过 Menu 菜单列出以上功能选择。实现的方法在第 5 章中已经讲解过，不知读者是否已经掌握，方法如下：

添加菜单：

```
public static final int GO_ID         = 1;
   public static final int BACK_ID     = 2;
   public static final int ZOOMOUT_ID = 3;
   public static final int ZOOMIN_ID  = 4;
   public static final int DEFAULT_ID = 5;
@Override
   public boolean onCreateOptionsMenu(Menu menu) {
      super.onCreateOptionsMenu(menu);
```

```
        //添加菜单项
        menu.add(0, GO_ID,       0, "前进");
        menu.add(0, BACK_ID,     1, "后退");
        menu.add(0, ZOOMOUT_ID,  2, "放大");
        menu.add(0, ZOOMIN_ID,   3, "缩小");
        menu.add(0, DEFAULT_ID,  4, "默认");
        return true;
}
```

实现按钮单击事件：在单击事件中，每个按钮的重要方法我们都已经有了讲解。这里值得一提的是在执行前进、后退时，需要先判断一下网页是否可以前进、后退。如果是第一个页面，那么回退到哪里去呢？

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case GO_ID:
        {
           if (wv.canGoForward())
           {
            wv.goForward();
           }
           return true;
        }
        case BACK_ID:
        {
         if (wv.canGoBack())
         {
             wv.goBack();
         }
            return true;
        }
        case ZOOMOUT_ID:
        {
           settings.setDefaultZoom(WebSettings.ZoomDensity.CLOSE);
           return true;
        }
        case ZOOMIN_ID:
        {
         settings.setDefaultZoom(WebSettings.ZoomDensity.FAR);
            return true;
        }
        case DEFAULT_ID:
        {
         settings.setDefaultZoom(WebSettings.ZoomDensity.MEDIUM);
            return true;
        }
    }
    return super.onOptionsItemSelected(item);
}
```

接着，重新运行代码，效果如图 11.4 所示。



登录界面



默认大小的新闻界面



放大后的网页



缩小后的网页

图 11.4　运行效果

11.4　小　　结

本章我们学习了如何使用 Android 进行 HTTP 连接，介绍了 HttpUrlConnection 以及 HttpClient 的使用，并使用 WebView 制作了一个自定义的浏览器。本章重点是 WebView 的使用，难点是 HTTP 连接中 GET 方法和 POST 方法的区别：POST 方法传递参数不能附加在 Url 中。下一章我们将学习使用 Android 中基于位置的服务——LBS（Location Based Service）。

第 12 章　Android 地图服务

在 Android 中嵌入的 Google 地图大家肯定不会陌生，地图服务已经是与我们日常生活密不可分的一个功能。使用 Google 为我们提供的 MapView，结合 MapActivity 可以轻松实现自定义的地图服务。本章将学习如何使用它完成地图显示、定位、搜索等一系列功能。

12.1　Google 地图显示

Google 为开发人员提供了许多强大的 API，我们只需调用这些 API 就可以快速地开发出想要实现的功能。本节将带领读者进入 GoogleMap 的开发世界，一起享受 LBS（Location Based Service）为我们带来的便利服务。学习的第一步就是将地图成功显示到手机上。

12.1.1　申请 Google Maps API 金钥

要在 Android 手机上开发 Google Map 地图服务，首先需要申请 Google Maps API 金钥，当然，这是免费的。该金钥可以在多个程序中使用，并不需要每个程序都申请一个。其申请流程如下：

（1）在本地获得 MD5 认证指纹。

（2）登录 Google 注册网页。

（3）使用指纹获得金钥。

接下来就开始第一步的工作，获得 MD5 认证指纹。

1．获得MD5认证指纹

如果您的环境是按照本书的指导安装的话，只需打开 Windows 命令行窗口，输入以下命令：

```
keytool -list -alias androiddebugkey -keystore "keystore路径" -storepass
android -keypass android
```

解释一下以上代码：androiddebugkey 表示该 debugkey 的别名；这里的 keystore 路径可以使用默认的 debugkey 的路径，在 Eclipse 中你可以找到它，方法如下：

（1）打开 Eclipse，选中 Window 菜单，在下拉菜单中选择首选项——Preference，如图 12.1 所示。

（2）在弹出的对话框中单击 Android|Build 选项，在右侧面板中的 Default debug keystore 栏中就可以找到具体的路径了，如图 12.2 所示。这里的路径如下：

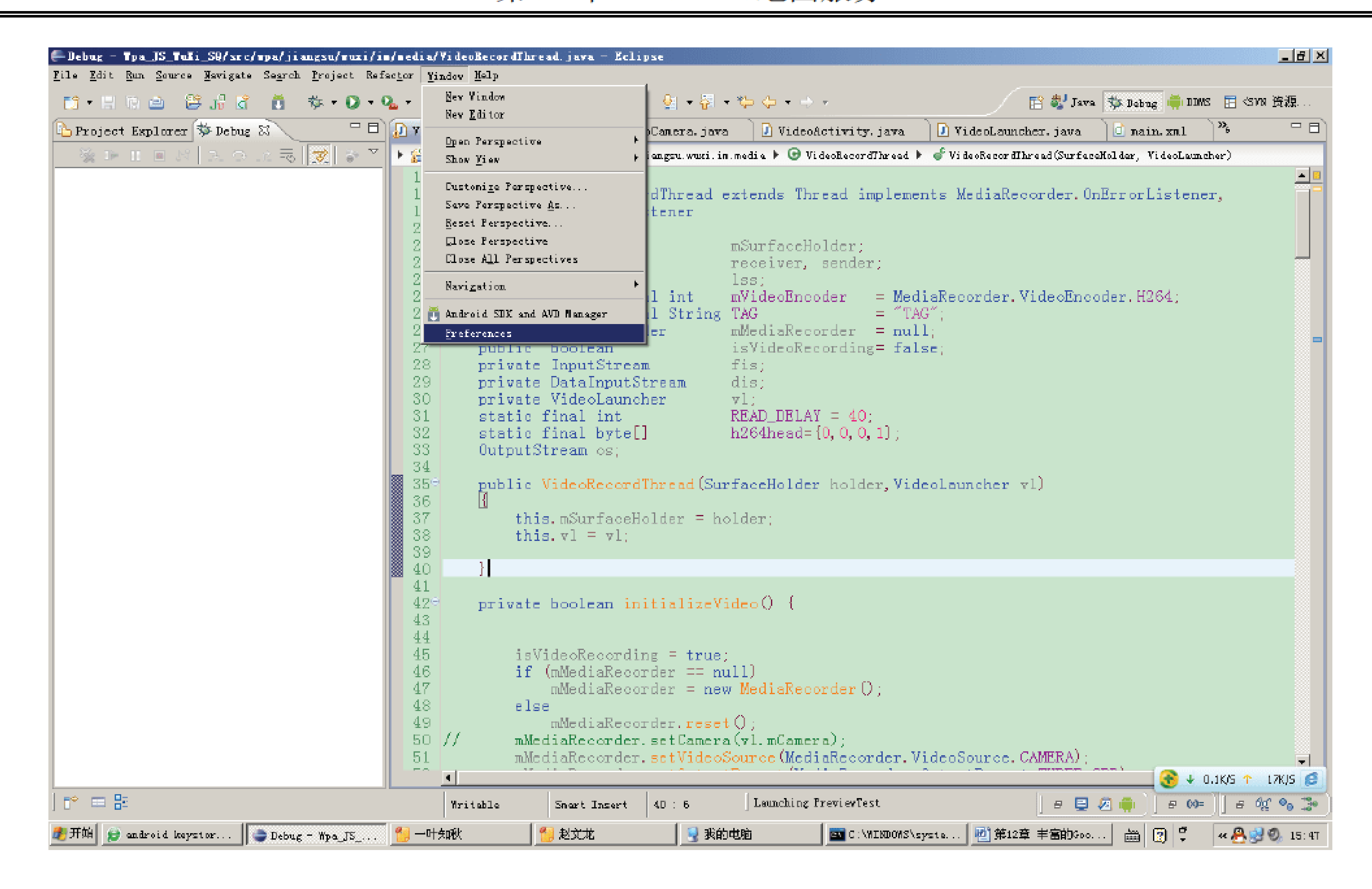


图 12.1　选择首选项

```
C:\Documents and Settings\Administrator\.android\debug.keystore
```

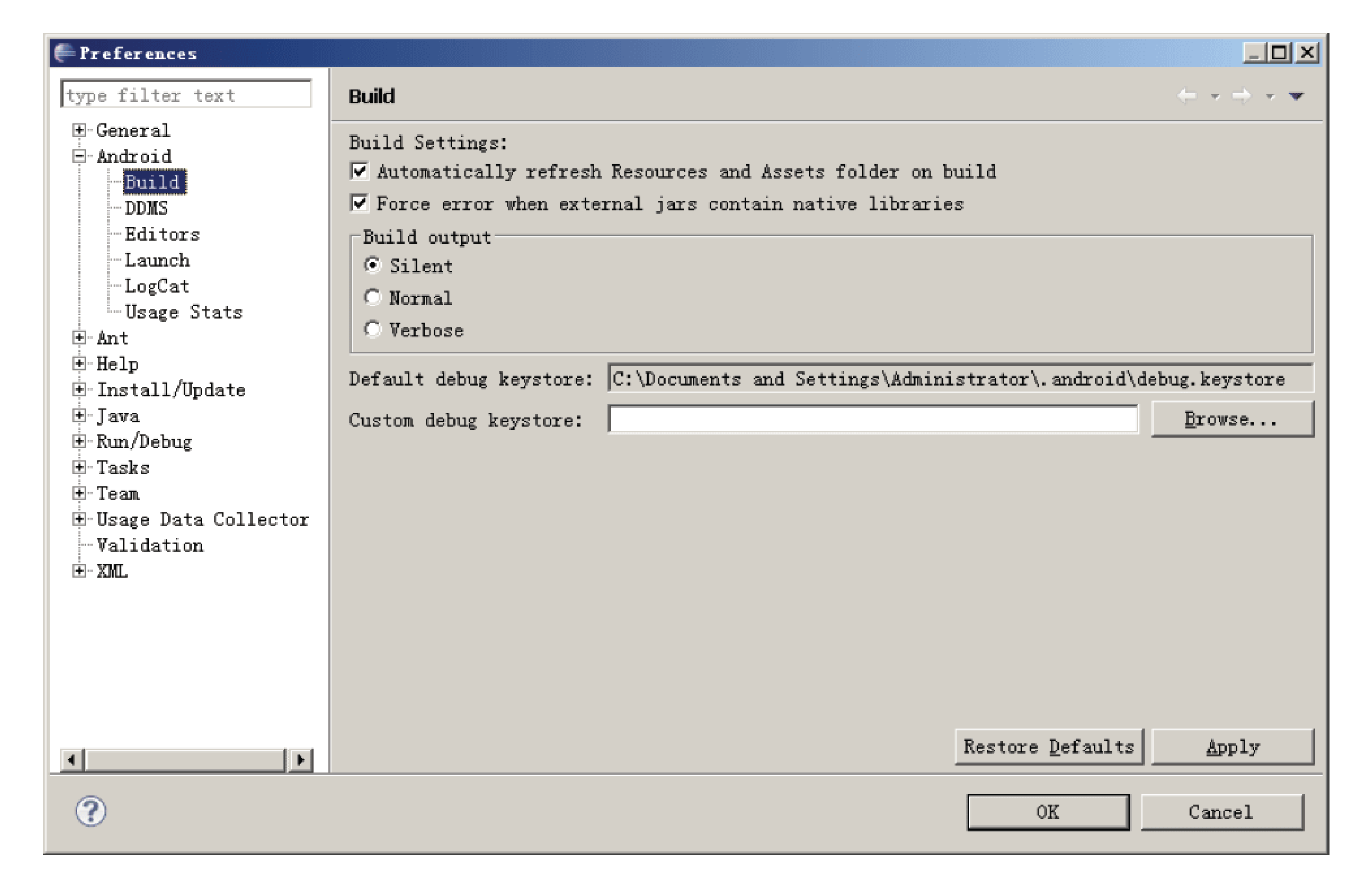


图 12.2　获得 Default debug keystore 路径

这样，完整的命令就是：

```
keytool -list -alias androiddebugkey -keystore " C:\Documents and Settings\Administrator\.android\debug.keystore " -storepass android -keypass android
```

运行成功后就可以得到本地的 MD 5 认证指纹了，如图 12.3 所示。

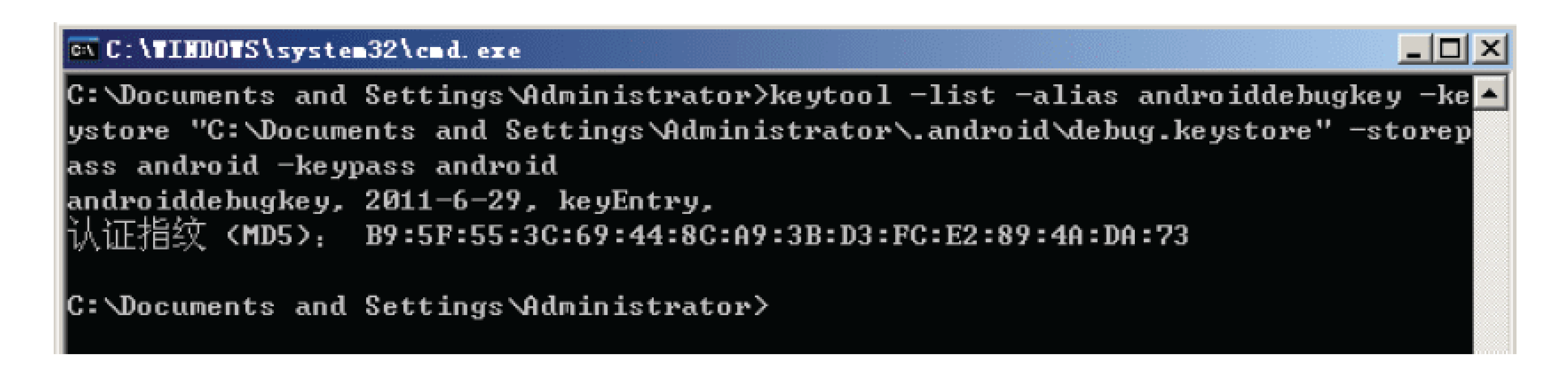


图 12.3　获得认证指纹

根据图 12.3 所示，本地的 MD5 认证指纹就是：

```
B9:5F:55:3C:69:44:8C:A9:3B:D3:FC:E2:89:4A:DA:73
```

如果没有成功生成认证指纹，那么请确定将 JAVA_HOMEd jre/bin 加入到了系统变量中，或者也可以先通过 cd 命令进入 jre/bin 目录下再重新尝试输入该命令。

2．登录Google注册网页

Google 注册网页的网址如下：http://code.google.com/intl/zh-CN/android/maps-api- signup.html，进入网页后显示如图 12.4 所示。

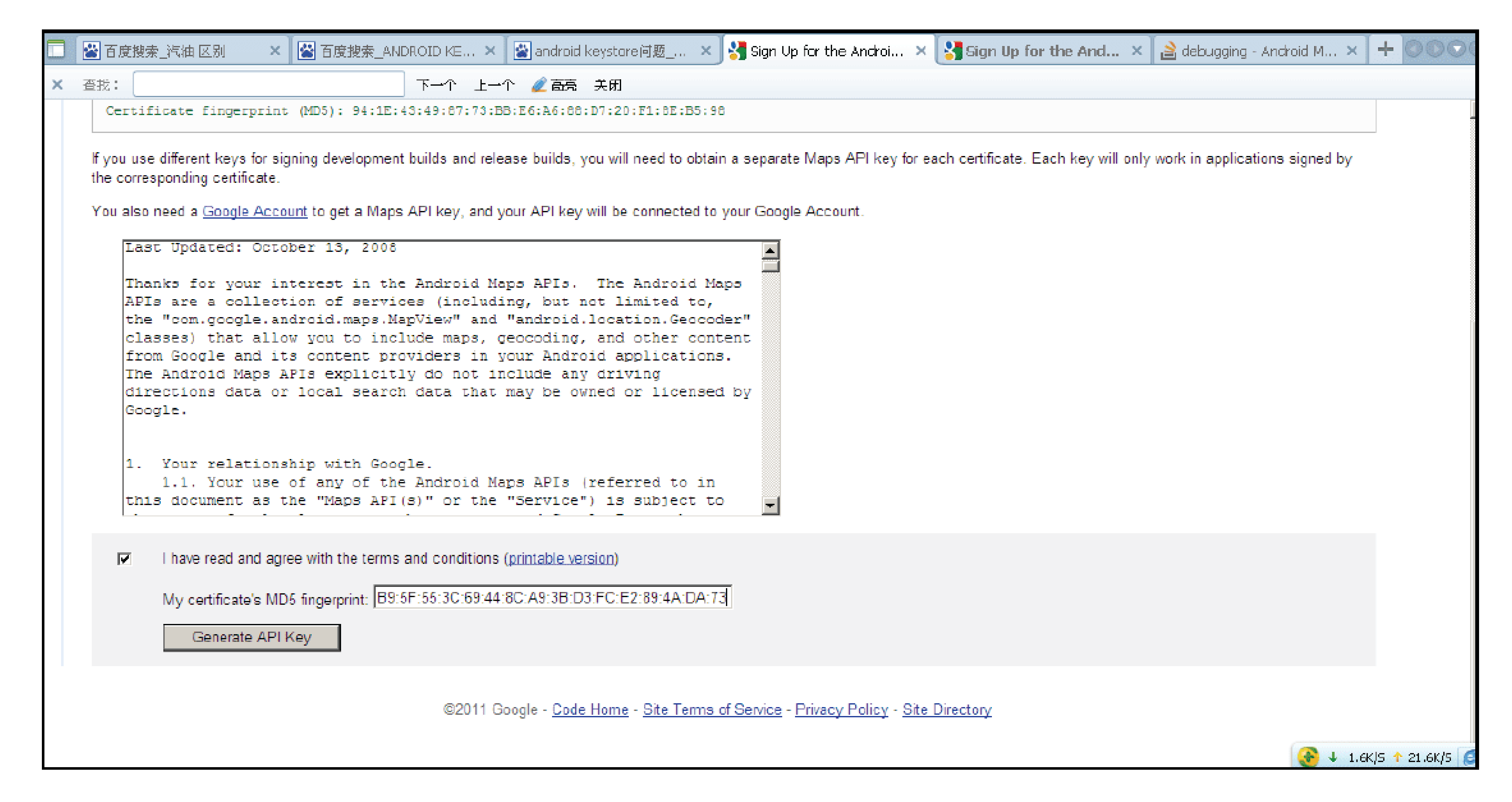


图 12.4　注册网页

3．生成金钥

在输入框中输入刚才获得的 MD5 认证指纹，单击 Generate API Key 按钮，生成了 Google Maps API 金钥，如图 12.5 所示。

如上所示，在 Google 的网页中也许会有一些乱码，不要管它，我们只需明确：第一个提示框中显示的就是 API 金钥了，这里的金钥是：

```
0nEgBgK-FjWmoXZap7PhTSm2PnjP-Gw1jTzUogQ
```

第二个提示框中显示的是 MD5 认证指纹，最后一个提示框中是一个 MapView 的简单

范例，我们会在下一小节进行详细讲解。至此，我们就完成了 Google Maps API 金钥的申请了，握着金钥，我们就可以开始开发 LBS 相关程序了。



图 12.5　获得金钥

12.1.2　使用 MapView 显示地图

本小节将讲解如何使用 MapView 进行 LBS 开发。使用 MapView 时必须与 MapActivity 结合起来一起使用，因为在 MapActivity 中它需要连接底层网络。接下来我们就着手进行地图的显示吧！步骤如下：

（1）在布局中添加 MapView 组件。

（2）在 Java 代码中获得其操作对象。

（3）通过 MapView 获得 MapController 对象。

（4）通过该 Controller 对象控制地图移动、放大等。

（5）在注册文件中添加权限以及使用包说明。

在之前申请 API 金钥的时候，网页中就已经给出了一个最简单的 MapView 的 xml 代码。

1．添加MapView组件

MapView 组件 xml 布局代码如下，注意一定要添加 apiKey 属性：

```
<com.google.android.maps.MapView
   android:id="@+id/mv"
   android:layout width="fill parent"
   android:layout height="fill parent"
   android:apiKey="0nEgBgK-FjWmoXZap7PhTSm2PnjP-Gw1jTzUogQ"
   />
```

这样 MapView 就配置完成了。

2．在Java代码中获得其操作对象

获得操作对象时，同样使用 Activity.findViewById(int id)方法，这一点就不再赘述。

3．通过MapView获得MapController对象

操作 MapView 时，我们通常使用 MapController 对象，获得该对象的方法是：

```
MapView.getController()
```

这样我们就获得了地图的控制器了，使用它可以进行移动地图、定位地图、放大及缩小地图等操作。

4．使用控制器

经常使用的一些控制器的方法如表 12-1 所示。

表 12-1　一些控制器的方法

方　法　名	作　　用
MapController.setCenter(GeoPoint point)	将地图中心定位到传入的参数位置
MapController.setZoom(int zoomLevel)	设置地图的缩放等级
MapController.zoomOut()	缩小地图
MapController.zoomIn()	放大地图

需要说明的是：这里的 setCenter()方法中传入的参数是 GeoPoint 类，这个类中包含了经度和纬度信息，可以将之理解为一个坐标点。获得该类对象的方法是：

```
GeoPoint.GeoPoint(int latitudeE6, int longitudeE6)
```

5．在注册文件中添加信息

由于开发 GoogleMap 我们需要联网，所以在注册文件中需添加使用网络的权限：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

在开发中我们使用了 Google 提供的 maps 库，所以还需要添加使用库的说明：

```
<uses-library android:name="com.google.android.maps"/>
```

最后添加了以上信息的注册文件显示如下：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.wes.mapdemo1"
     android:versionCode="1"
     android:versionName="1.0">
   <uses-sdk android:minSdkVersion="8" />

   <application android:icon="@drawable/icon" android:label="@string/
   app name">
      <activity android:name=".MapDemo1"
              android:label="@string/app name">
          <intent-filter>
             <action android:name="android.intent.action.MAIN" />
             <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
      </activity>
    <uses-library android:name="com.google.android.maps"/>
   </application>
   <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

12.1.3　通过实例使用 MapView

本小节中我们将通过代码写出一个自己的 GoogleMap，地图只需包含一些基本的使用功能：地图展示、地图移动、放大缩小以及切换不同的视图比如图街景视图、交通视图以及卫星视图。接下来就开始我们的实例吧。

首先新建一个工程，注意这里需要选择使用的 SDK target 为 Google APIs Platform 2.2。否则我们将无法使用 Google 地图，如图 12.6 所示。

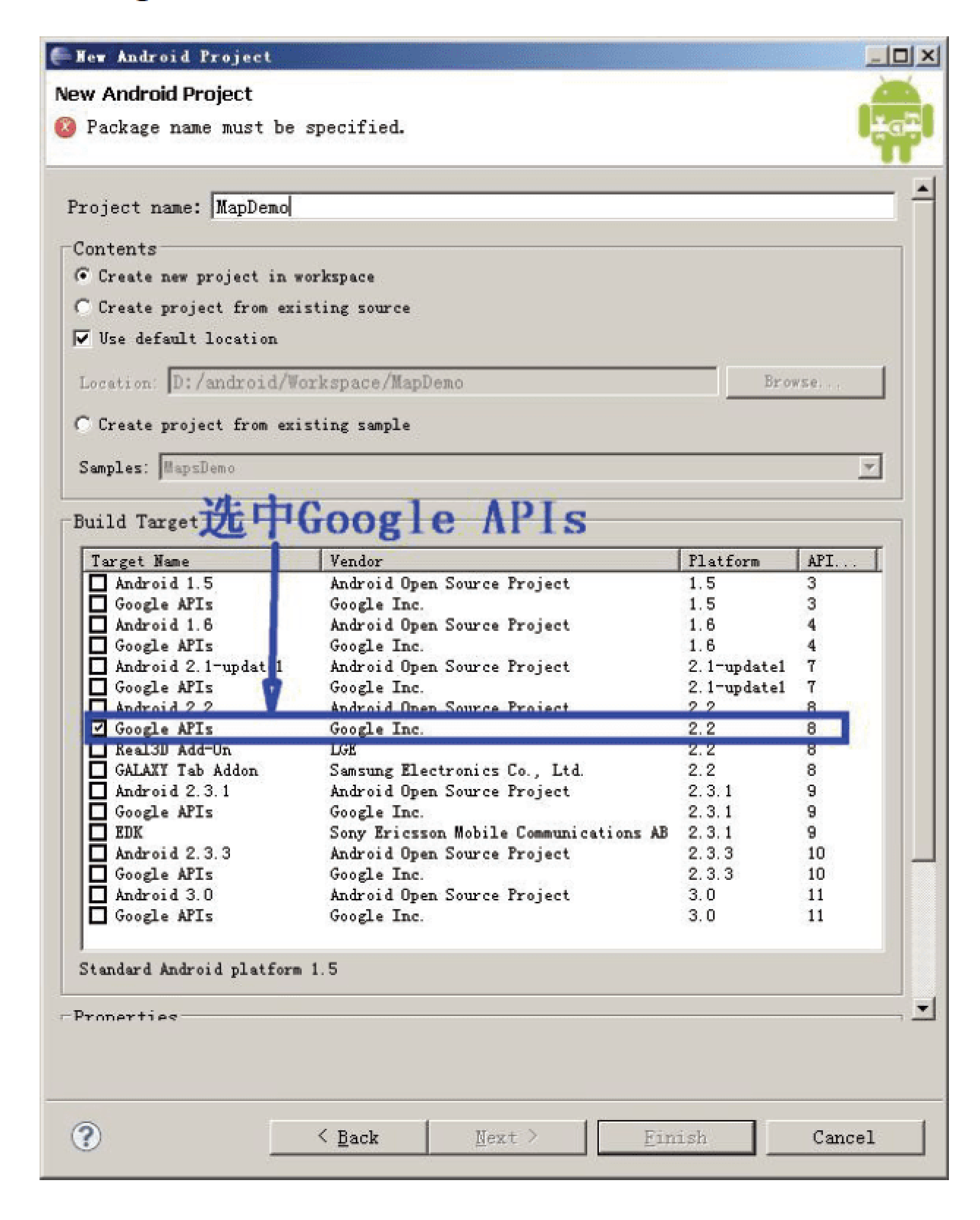


图 12.6　选择 Google APIs

1．xml代码

在布局文件需要添加组件如表 12-2 所示。

表 12-2　MapDemo1 中布局文件的组件

类　　型	ID	意　　义
MapView	Mv	地图显示组件
Button	East	单击后地图向东移动
Button	West	单击后地图向西移动
Button	North	单击后地图向北移动
Buton	South	单击后地图向南移动
Button	Zoomin	单击后放大地图
Button	Zoonout	单击后缩小地图

如何将以上组件组织起来就由读者朋友们随意发挥了，这里使用的是框架布局 FrameLayout，在上、下、左、右各放一个按钮，底部显示“放大”、“缩小”按钮。

2. Java代码

在 Java 代码中，我们首先进行整体设计：

```
import ......                //省略部分导入
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;

public class MapDemo1 extends MapActivity {
    MapView mv;
    MapController controller;
    Button b east;
    Button b north;
    Button b west;
    Button b south;
    Button b zoomin;
    Button b zoomout;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        initView();                  //初始化视图
        //得到 MapView 的控制器
        controller = mv.getController();
        //设置地图缩放等级，参数值在 1～21 之间
        controller.setZoom(10);
}
}
```

接着，我们实现按钮的单击监听事件：

```
public void initView()
{
    mv        = (MapView) findViewById(R.id.mv);
    b_east    = (Button) findViewById(R.id.east);
    b_west    = (Button) findViewById(R.id.west);
    b_south   = (Button) findViewById(R.id.south);
    b_north   = (Button) findViewById(R.id.north);
    b_zoomin  = (Button) findViewById(R.id.zoomin);
    b zoomout= (Button) findViewById(R.id.zoomout);

     OnClickListener l = new OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            int id = v.getId();
            switch(id)
            {
            case R.id.east:
            {
                panEast();                  //向东移动
```

```
                break;
            }
            case R.id.west:
            {
                panWest();                  //向西移动
                break;
            }
            case R.id.north:
            {
                panNorth();                 //向北移动
                break;
            }
            case R.id.south:
            {
                panSouth();                 //向南移动
                break;
            }
            case R.id.zoomin:
            {
                zoomIn();                   //放大地图
                break;
            }
            case R.id.zoomout:
            {
                zoomOut();                  //缩小地图
                break;
            }
            }
        }
    };
    //设置监听事件
    b_east.setOnClickListener(l);
    b_west.setOnClickListener(l);
    b_north.setOnClickListener(l);
    b_south.setOnClickListener(l);
    b_zoomin.setOnClickListener(l);
    b_zoomout.setOnClickListener(l);
}
```

现在整体框架已经搭建完毕，接下来就是真正的功能实现了，首先是移动地图的功能，我们以向东移动为例：

```
public void panEast()
    {
        //得到经度值
        int latitude  = mv.getMapCenter().getLatitudeE6();
        //得到纬度值
        int                          longitude                          =
mv.getMapCenter().getLongitudeE6()+mv.getLongitudeSpan()/3;
        //通过经纬度得到坐标点
        GeoPoint point = new GeoPoint(latitude, longitude);
        //将地图中心设置为坐标点
        controller.setCenter(point);
    }
```

首先得到地图当前的中心点：

```
MapView.getMapCenter()
```

该方法返回的是 GeoPint 对象，接着从 GeoPoint 对象中提取经度值和纬度值，方法分别为：

```
GeoPoint.getLatitudeE6()
GeoPoint.getLongitudeE6()
```

我们知道纵向的是经度，横向的是纬度，向东移动时，经度值不变，纬度值增加就可以了。那么增加多少呢？是自己写一个 int 值吗？答案是可以，但不好控制，一般情况下我们首先得到当前地图的跨度。例如，得到纬度上的跨度，也就是当前显示的地图从左边沿到右边沿之间的距离，方法为：

```
MapView.getLongitudeSpan()
```

所以要向东移动 1/3 个屏幕大小，纬度值就增加 MapView.getLongitudeSpan()/3。

同理，向西移动，纬度值就减少 MapView.getLongitudeSpan()/3。

接着通过新的经度值和纬度值得到新的 GeoPoint 点：

```
GeoPoint.GeoPoint(int latitudeE6, int longitudeE6)
```

最后，将地图的中心定位到新的坐标点就实现向东移动功能了。

聪明的读者肯定已经能够写出向南、北移动的方法了吧。这里就不再详细讲解，直接给出其余 3 个方向的方法：

```
public void panWest()
    {
        //向西减少纬度
        int latitude  = mv.getMapCenter().getLatitudeE6();
        int longitude = mv.getMapCenter().getLongitudeE6()-mv.
        getLongitudeSpan()/3;
        GeoPoint point = new GeoPoint(latitude, longitude);
        controller.setCenter(point);
    }

    public void panSouth()
    {
        //向南减少经度
        int latitude  = mv.getMapCenter().getLatitudeE6() - mv.
        getLatitudeSpan()/3;
        int longitude = mv.getMapCenter().getLongitudeE6();
        GeoPoint point = new GeoPoint(latitude, longitude);
        controller.setCenter(point);
    }

    public void panNorth()
    {
//向北增加经度
        int latitude  = mv.getMapCenter().getLatitudeE6()+mv.
        getLatitudeSpan()/3;
        int longitude = mv.getMapCenter().getLongitudeE6();
        GeoPoint point = new GeoPoint(latitude, longitude);
        controller.setCenter(point);
    }
```

接着实现放大、缩小的功能，代码非常简单，只需一行代码就搞定了：

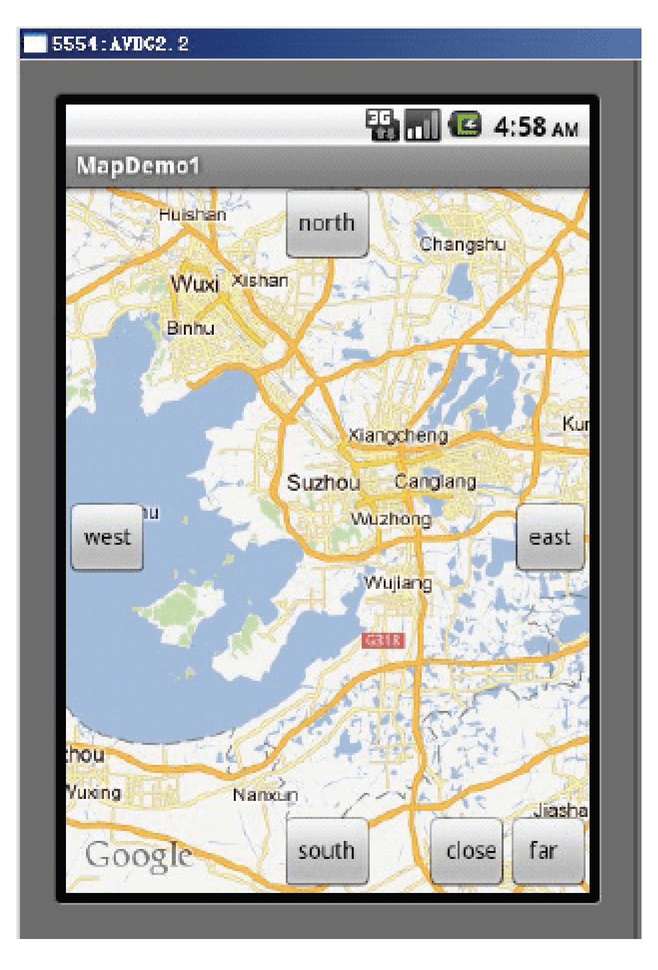
```
public void zoomOut()
```

```
    {
        controller.zoomOut();
    }

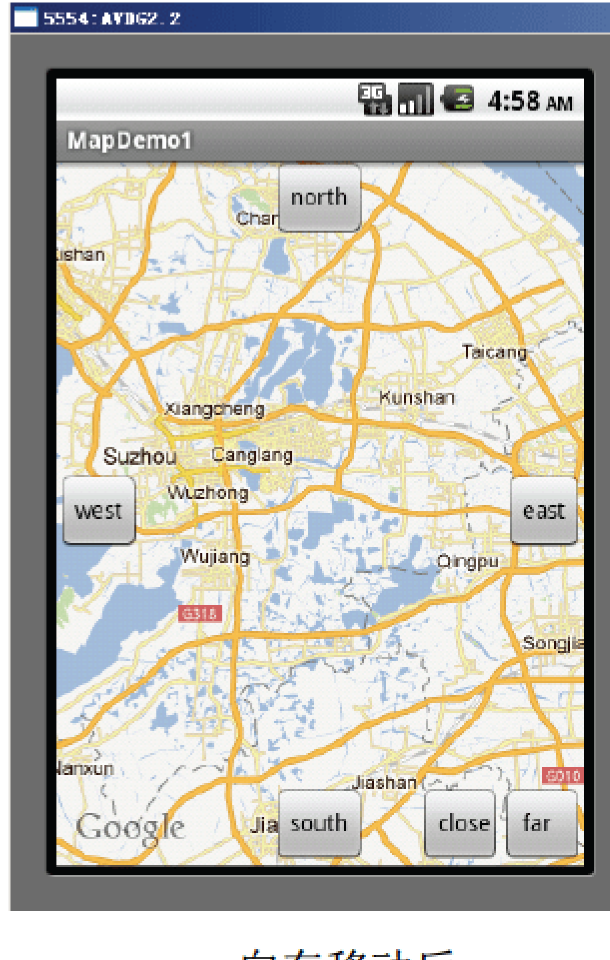
    public void zoomIn()
    {
        controller.zoomIn();
    }
```

现在看来是不是发现地图开发也没有想象中那么困难呢？

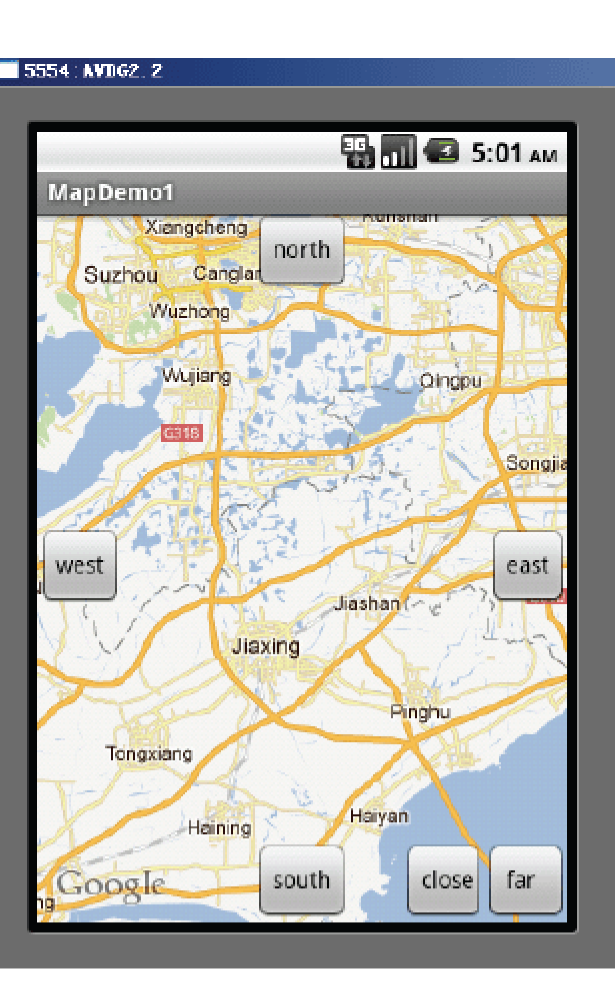
现在来运行一下程序，我们会看到模拟器上成功显示了地图，首先尝试一下移动地图，如图 12.7 所示。



地图显示



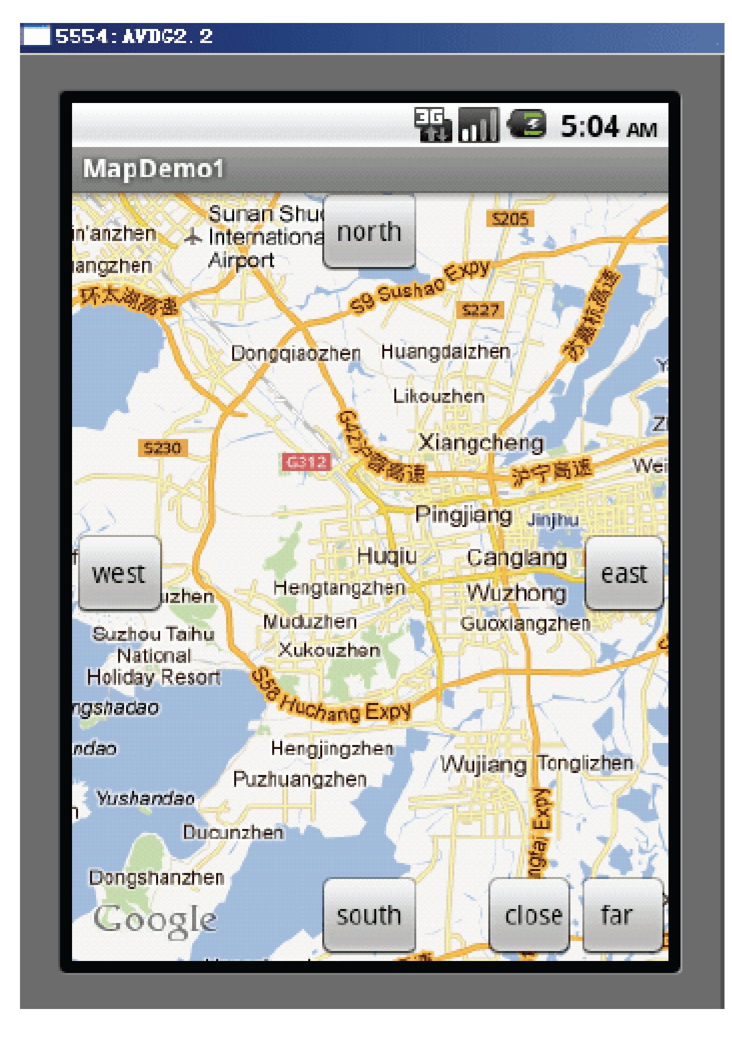
向东移动后



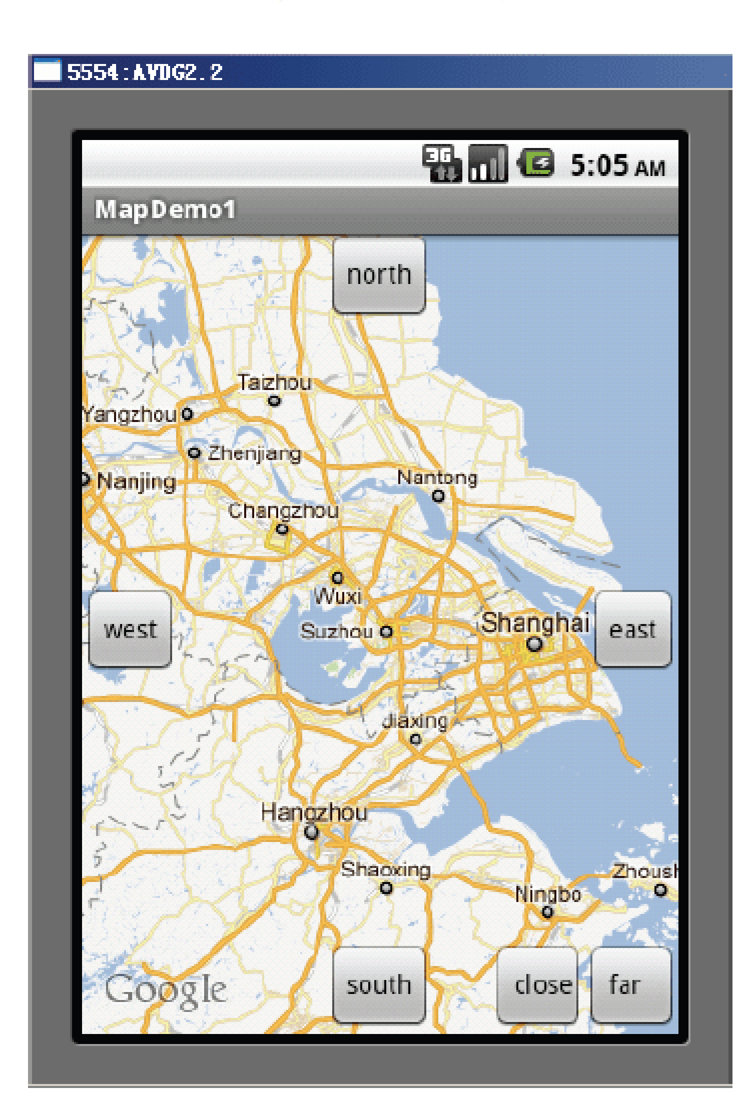
向北移动后

图 12.7　移动地图

接下来我们可以尝试一下放大、缩小功能是否可以正常使用，如图 12.8 所示。



放大后地图显示



缩小后地图显示

图 12.8　放大、缩小地图

到这里我们的地图基本功能已经实现。当然我们还有更多强大的功能没有使用，如我

们可以切换一下视角，看一下卫星显示图，或者看一下街景视图。现在继续在地图上添加组件已经不合适了，太多的组件会给用户操作带来不好的感觉，我们可以将该功能放到 Menu 中，方法如下：

```
public static final int STREET_ID  = 1;
   public static final int TRAFFIC_ID = 2;
   public static final int SATE_ID    = 3;

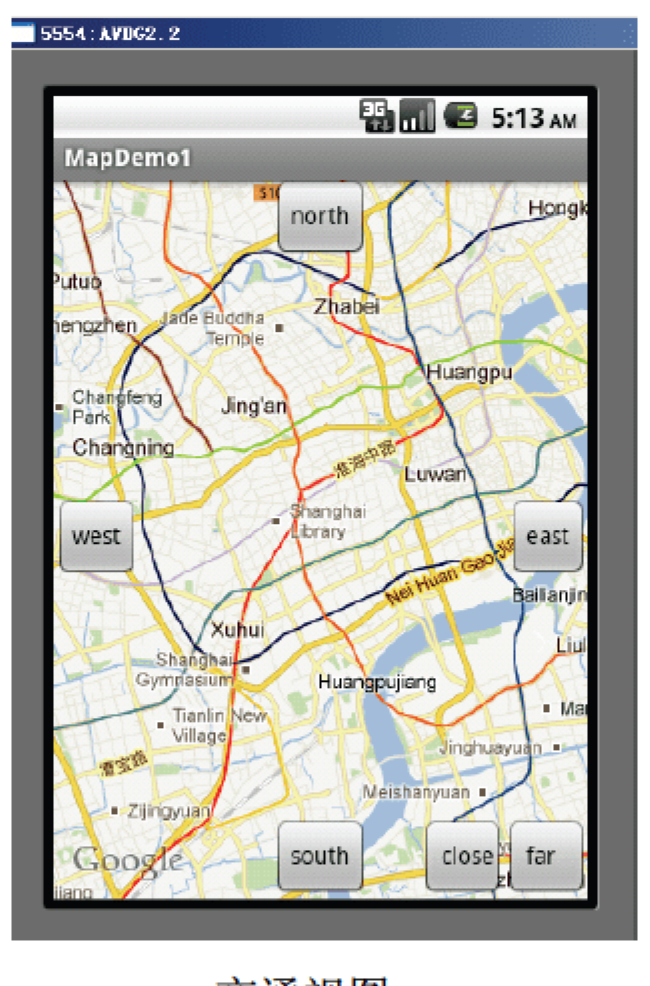
@Override
   public boolean onCreateOptionsMenu(Menu menu) {
      super.onCreateOptionsMenu(menu);
      //将选项添加到菜单中
      menu.add(0, TRAFFIC_ID, 0, "交通");
      menu.add(0, SATE_ID,    1, "卫星");
      menu.add(0, STREET_ID,  2, "街道");
      return true;
   }

   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
          case TRAFFIC_ID:
          {
             //设置交通视图
            mv.setTraffic(true);
            mv.setSatellite(false);
            mv.setStreetView(false);
            return true;
          }
          case SATE_ID:
          {
//设置卫星视图
           mv.setSatellite(true);
           mv.setStreetView(false);
           mv.setTraffic(false);
             return true;
          }
          case STREET_ID:
          {
             //设置街景视图
            mv.setStreetView(true);
            mv.setSatellite(false);
            mv.setTraffic(false);
            return true;
          }

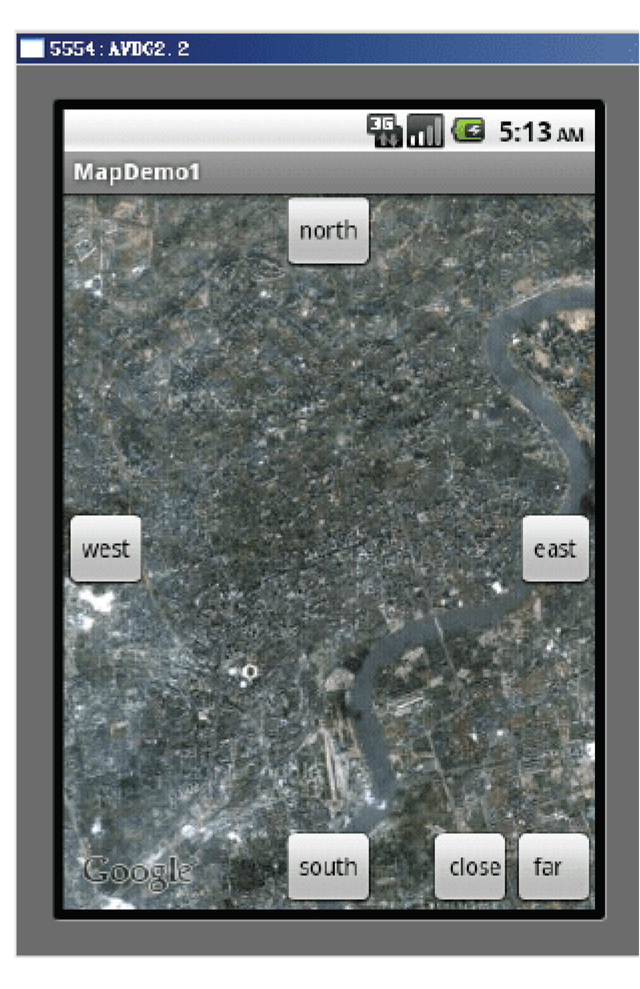
      }
      return super.onOptionsItemSelected(item);
   }
```

此时单击 MENU 按钮，我们就可以切换视图模式了，如图 12.9 所示。

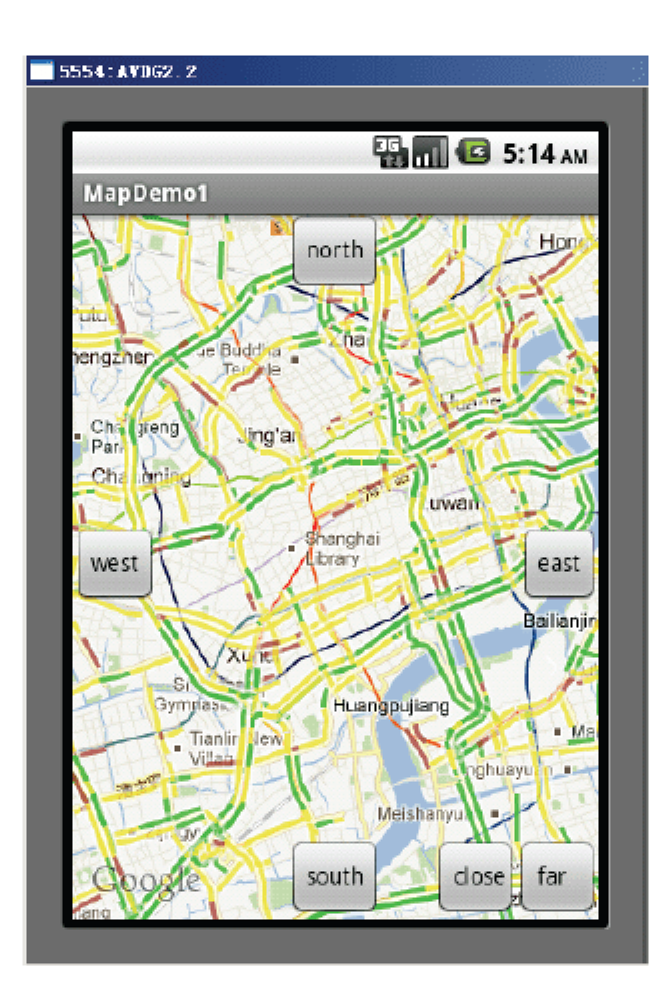
也许你觉得单击按钮移动地图不方便，因为通常我们使用地图时都是拖曳的。该功能是不是很难实现呢？其实不然，只需添加一条属性就可以了：



交通视图



卫星视图



街景视图

图 12.9　显示不同视图

```
<com.google.android.maps.MapView
    android:id="@+id/mv"
    android:clickable="true"
    android:layout width="fill parent"
    android:layout height="fill parent"
    android:apiKey="0nEgBgK-FjWmoXZap7PhTSm2PnjP-Gw1jTzUogQ"
    />
```

添加加粗部分的属性，我们就可以顺利地拖曳视图了，这样，我们就可以把地图上那些东、西、南、北的按钮全部去掉，这样界面就更简洁了。更多强大的功能还需读者自己去发掘，下一节我们将讲解使用 GPS 定位我的位置。

12.2　使用 GPS

本节我们将学习如何使用 GPS（Global Positioning System）。在使用地图时，我们必须使用的一个功能就是确定自己的位置。那么在 Android 中如何实现获得自己的位置呢？如果位置改变了如何更新呢？这些问题的答案，在本节中我们将一一给出。

12.2.1　获得我的位置

在 Android 中如果要获得自己所在的位置，必须要执行一些必要的操作。其步骤如下：

（1）获得位置管理器——LocationManager。

（2）通过一系列的条件，选择服务提供商。

（3）实现 LocationListener 类。

（4）启动监听位置信息的变化。

（5）在注册文件中添加必要的权限。

接下来进入每个步骤的详细讲解。

1．获得位置管理器

位置管理器是从 SystemService 中获得的，这是最基础的，也是最重要的，它相当于这个地图服务的大脑，管理着一系列的工具，比如位置信息提供商以及位置信息监听器等。获得的方法如下：

```
LocationManager manager = (LocationManager) getSystemService(Context.
LOCATION_SERVICE);
```

2．获得位置信息提供商

通过管理器，我们可以得到位置信息的提供商，不过在获得提供商之前，我们需要进行一下筛选，通过一些条件选择出一个最合适的提供商。具体的方法如下：

（1）新建一个 Criteria 对象

该对象的作用是设置一些选择的依据，如精确度、耗电等级等，获得的方法为：

```
Criteria cri = new Criteria();
```

（2）设置必要的条件

使用该对象的一些方法可以设置我们需要的条件，常用的设置方法如表 12-3 所示。

表 12-3　常用的条件选择

方　法　名	意　　义
Criteria.setAccuracy(int accuracy)	该方法用于设置获得的位置信息的精确度，可以设置为 Criteria.ACCURACY_FINE 或者 ACCURACY.COARSE，这两个参数分别表示精确和粗略
Criteria.setPowerRequirement(int level)	设置耗电等级，可以设置为高级、中级和低级，分别对应着： Criteria.POWER_HIGH Criteria.POWER_MEDIUM Criteria.POWER_LOW
Criteria.setCostAllowed(boolean costAllowed)	设置是否允许向用户收费
Criteria.setAltitudeRequired(boolean altitudeRequired)	设置是否要得到海拔信息
Criteria.setBearingRequired(boolean bearingRequired)	设置是否要得到方位信息

需要注意的是，并不是只要设置了条件之后，提供商就一定能提供相应的信息，管理器只是获得一个最合适的提供商而不是一个完全符合条件的提供商。

（3）通过条件获得提供商

获得提供商的方法为：

```
LocationManager.getBestProvider(Criteria criteria, boolean enabledOnly)
```

该方法返回的是一个提供商的名称，是 String 类型的。

3. 实现位置监听器

在第二步中我们得到了提供商，可是仅仅只有提供商为我们提供的位置信息，我们本地不接收的话，那一切岂不是无用功么？所以，在我们的程序中需要新建一个 Location Listener 类，用以监听位置的变化。方法如下所示：

```
LocationListener locationListener = new LocationListener()
   {
    @Override
    public void onLocationChanged(Location location)
    {
         //位置改变时要进行的操作
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras)
    {
        //状态改变时要进行的操作

    }

    @Override
    public void onProviderEnabled(String provider)
    {
        //服务可用时进行的操作
    }

    @Override
    public void onProviderDisabled(String provider)
    {
        //服务不可用时进行的操作
    }
   };
```

实现 LocationListener 时，需要重写它的 4 个方法，分别表示位置改变、状态改变、服务是否可用。其对应的方法名在程序中已经给出了注释，这里就不再一一列举。

4. 启动位置监听

到目前为止，我们已经拥有了提供商，拥有了监听装置，那么接下来要做的工作就是将它们有机地结合起来，实现强大的功能。此时就需要 LocationManager 了，因为它充当着大脑的角色。具体的方法是：

```
LocationManager.requestLocationUpdates(String  provider,  long  minTime,
float minDistance, LocationListener listener)
```

该方法拥有 3 个参数，它们分别是：

（1）String provider：位置信息提供商的名字。

（2）long minTime：最小的更新时间。

（3）float minDstance：最短的更新距离。

（4）LocationListener listener：位置监听器。

这样一来，一旦位置信息发生改变，我们就能够及时地了解并进行相关的操作了。

5．在注册文件中添加权限

最后，我们还需要在注册文件中添加位置权限以及网络权限。具体代码如下：

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

这里的两个位置权限分别代表了精确的位置信息和粗略的位置信息。这样，我们就成功地完成了位置变化的监听了。

12.2.2　通过实例完成 GPS 的使用

本小节将通过一个实例完成 GPS 功能的使用。本实例要实现的功能是：（1）显示地图并允许拖曳；（2）及时地得到我的位置，将地图定位到该坐标点。接下来，我们就开始新建工程。再次强调，开发 GoogleMap 程序时，必须使用 Google Mpas API，否则程序无法正常运行。

1．xml代码

在 xml 代码中，我们主要需要添加两部分组件，一部分是 MapView；另一部分就是 3 个按钮，分别用来实现不同的功能。

（1）添加 MapView 组件

在添加 MapView 组件时，千万不能忘记添加 apiKey 属性。同时，为了使用方便，我们将 clickable 属性设置为“true”，这样地图就可以拖曳了。最后的范例代码如下：

```
<com.google.android.maps.MapView
    android:id="@+id/mv"
    android:clickable="true"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:apiKey="0nEgBgK-FjWmoXZap7PhTSm2PnjP-Gw1jTzUogQ"
    />
```

（2）添加按钮

本例中一共添加了 3 个按钮，分别用于放大地图、缩小地图和返回到我的位置，如表 12-4 所示。

表 12-4　MapDemo2 中按钮列表

类　型	ID	意　　义
Button	Zoomin	放大地图
Button	Zoomout	缩小地图
Button	Myaddr	返回到我的位置

2．Java代码

在 Java 代码中，我们依然首先进行整体的设计。

（1）整体设计

```
import com.google.android.maps.GeoPoint;
```

```
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import ......                              //此处省略部分导入

public class MapDemo2 extends MapActivity {
    MapView mv;
    MapController controller;
    LocationManager manager;
    String locationProvider;
    public LocationListener locationListener;
    GeoPoint curPoint;
    Button b zoomin;
    Button b zoomout;
    Button b myaddr;


    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        initView();
        //得到 LocationManager 对象
        manager = (LocationManager) getSystemService(Context.LOCATION
        SERVICE);
        //得到提供商
        locationProvider = getLocationProvider(manager);
        //新建位置监听器
        locationListener = new LocationListener()
        {
         @Override
         public void onLocationChanged(Location location)
         {
              //从 Location 中获得 GeoPoint 对象
              curPoint = getGeoPoint(location);
              //将地图移动到该地点
              controller.animateTo(curPoint);
         }

         @Override
         public void onStatusChanged(String provider, int status, Bundle extras)
         {

         }

         @Override
         public void onProviderEnabled(String provider)
         {

         }

         @Override
         public void onProviderDisabled(String provider)
         {
```

```
            }
        };
        //开始监听位置变化信息
        manager.requestLocationUpdates(locationProvider, 1000, 10,
        locationListener);
    }


    @Override
    protected boolean isRouteDisplayed()
    {
        return false;
    }
}
```

通过阅读以上代码，我们很清晰地发现这是完全按照 12.2.1 小节中讲解的步骤进行的。让我们重新整理这 4 个步骤：

- 获得位置管理器——LocationManager。
- 通过一系列的条件，选择服务提供商。
- 实现 LocationListener 类。
- 通过 requestLocationUpdates()方法启动监听位置信息的变化。

（2）实现 getLocationProvider()方法

通过 Criteria 对象设置选择条件，通过 LocationManager.getBestProvider(Criteria criteria, boolean enabledOnly)方法获得最合适的信息提供商，代码如下：

```
public String getLocationProvider(LocationManager lm)
    {
        //新建一个选择提供商的标准
        Criteria cri = new Criteria();
        //设置精确度为精确，还可以设置为粗略 ACCURACY.COARSE
        cri.setAccuracy(Criteria.ACCURACY FINE);
        //设置耗电等级为低
        cri.setPowerRequirement(Criteria.POWER_LOW);
        //设置是否向用户收费
        cri.setCostAllowed(true);
        //设置是否需要海拔信息
        cri.setAltitudeRequired(false);
        //设置是否需要方位信息
        cri.setBearingRequired(false);
        //得到最好的内容提供商
        String lp = lm.getBestProvider(cri, true);
        return lp;
    }
```

（3）实现 getGeoPoint()方法

在 LocationListener 类中，onLocationChanged()回调函数时，会传入一个 Location 对象，从这个对象中我们可以得到经度信息和纬度信息，得到了经纬度之后我们就能新建 GeoPoint 对象，接着我们就可以调用 MapController.animateTo(GeoPoint point)方法将地图移动到该点了。具体的代码为：

```
public GeoPoint getGeoPoint(Location location)
    {
        //得到经度
```

```
        double latitude = location.getLatitude()*1E6;
        //得到纬度
        double longitude= location.getLongitude()*1E6;
        //得到 GeoPoint 对象
        GeoPoint point = new GeoPoint((int)latitude,(int)longitude);
        return point;
    }
```

接着我们就可以着手调试程序了，注意使用模拟器调试程序时需要给出模拟的经纬度信息。方法如下：

进入 Eclipse 中的 DDMS，选中需要调试的机器，在下部面板的 Emulator Control 下选择 Location Controls，在该编辑框中就可以输入经纬度信息了，如图 12.10 所示。

编辑好经纬度信息后，单击 Send 按钮就可以发送了，在我们的程序中就会收到位置信息的变化，从而改变地图的显示了，程序运行后效果如图 12.11 所示。注意更新位置信息时，需要以 1 秒为周期，否则程序可能会不作响应。因为在 requestLocationUpdates()方法被调用时，我们设定的最小更新周期为 1 秒。

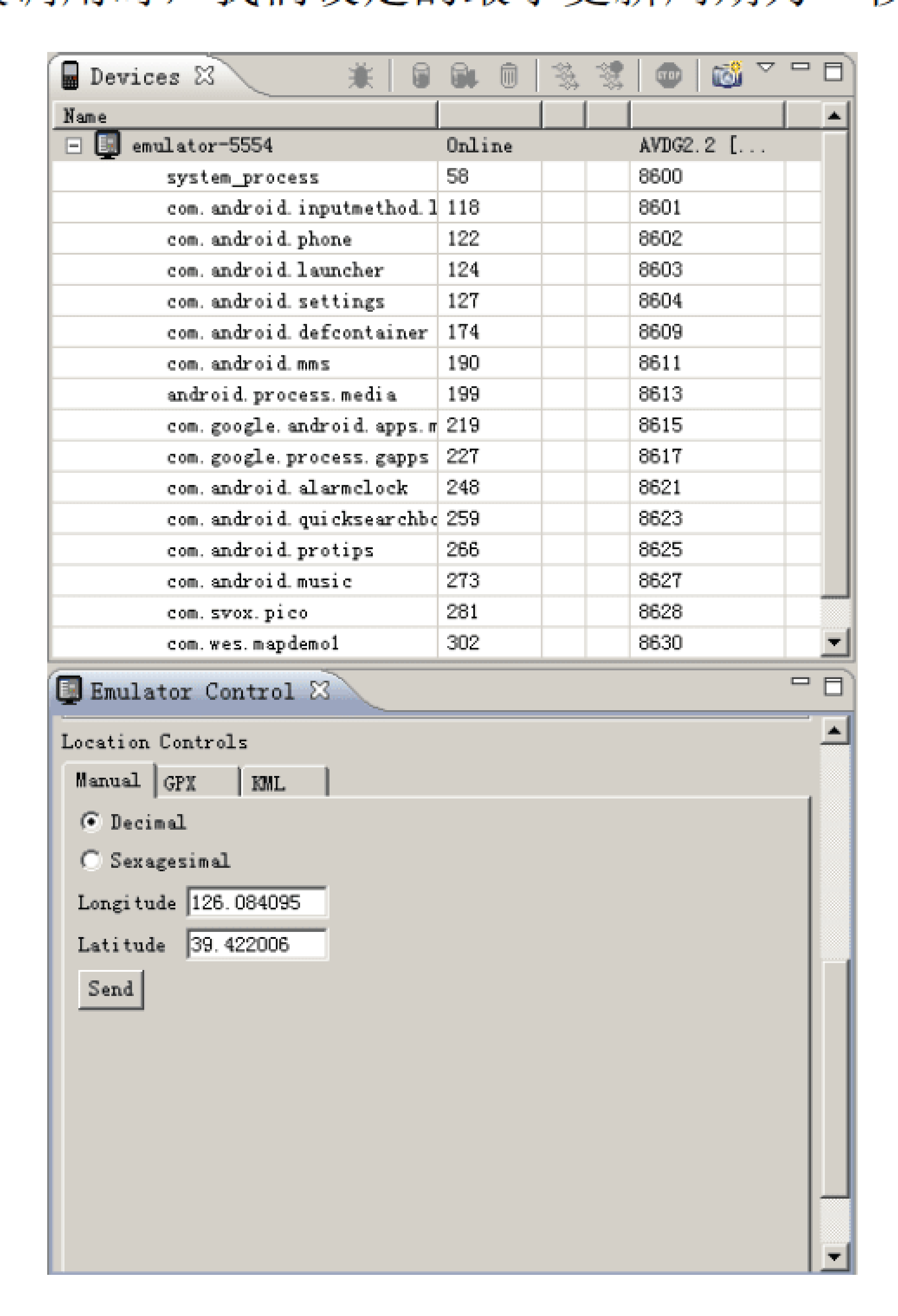


图 12.10　模拟位置信息



图 12.11　更新我的位置

12.3　使用地理位置编码

本节中要讲解的是使用地理位置编码。在程序中如果直接向用户显示经纬度信息非常不友好，这时如果将经纬度转换为实际的地址效果会比较好。Android 为我们提供了 Geocoder 类，以实现地址名称与经纬度值的转换。

12.3.1　转换地址信息

一般情况下，用户并不需要看到精确的经纬度的值，所以在得到经纬度以后，我们需要将之转换为实际的地址。在转换地址信息时又分为两类：

- 将得到的经纬度值转换为实际的地址。
- 将用户输入的地址转换为经纬度，并在地图上进行定位。

接下来我们首先讨论第一种情况。

1. 将经纬度转换为实际地址

转换位置时需要如下 3 个步骤：

（1）得到 Geocoder 对象。

（2）通过 Geocoder.getFromLocation()方法得到地址对象。

（3）操作返回的地址对象。

只需简单的 3 步操作，我们就可以将一个看起来没有头绪的经纬度值转换为一个可以被直观认识的地址信息了。这一切都要从获得 GeoCoder 对象开始。

（1）获得 Geocoder 对象

Geocoder 可以帮助我们处理地址编码并进行转换。要获得该对象并不是那么麻烦，其构造方法为：

```
Geocoder.Geocoder(Context context, Locale locale)
```

（2）得到地址列表

拥有了 Geocoder 对象后，我们就可以开始转换地址信息了，只需调用如下方法：

```
Geocoder.getFromLocation(double latitude, double longitude, int maxResults)
throws IOException
```

这里有 3 个参数，第一、二个是经度值和纬度值大家肯定不会陌生，第三个参数是最大结果。因为一个地点可能会返回多个查询结果，这里就限定了最大的结果值。该查询返回的值是一个最符合猜测的值，并不保证是一个精确的值。该方法的返回值是 List<Address>对象。

（3）处理得到的地址信息

在第二步得到的列表中，我们需要进一步处理。首先从中取出 Address 对象，接下来处理时常用的方法，如表 12-5 所示。

表 12-5　处理地址时常用的方法

方　法　名	作　　用
Address.getMaxAddressLineIndex()	得到地址最大行的索引
Address.getAddressLine(int index)	得到地址名
Address.getPostalCode()	得到邮政编码
Address.getCountryName()	得到国家名
Address.getPhone()	得到电话

2．将实际地址转换为经纬度

将实际地址转换为经纬度时，第一、二步与之前的操作相同，不同的是在得到了 Address 后需要进行的操作。操作时需通过以下方法得到 Address 对象中包含的经度值和纬度值：

```
Address.getLatitude()
Address.getLongitude()
```

得到值后不要忘记乘以 1E6，这样才能正确地在我们的地图上显示。

最后将经纬度值作为参数新建一个 GeoPoint 对象，这一步想必读者朋友们应该没有问题了。

12.3.2　通过实例使用地理位置编码

本小节将通过一个实例完成对地理位置编码的使用。该实例主要的功能包括：

（1）输入地名执行查询，并将地图定位到该点。

（2）实时更新我的位置，并在位置改变时显示我现在的位置信息。

首先，新建一个工程，不过在选择 SDK 时，请选择 Google Maps API Level 7，这是因为 2.2 版本的 Google Maps API 在使用时 Geocoder 存在 BUG，会捕获 Service not available 异常，导致无法使用。所以我们只能暂时将版本改为 2.1。

1．xml代码

在 xml 中，我们添加组件如表 12-6 所示。

表 12-6　MapDemo2 中的组件

类　　型	ID	作　　用
MapView	Mv	显示地图
TextView	Tv	显示目前的实际地理位置名称
EditText	Et	用来输入要查询的地址
Button	Search	单击后开始查询输入的地址名，并将地图定位到该点
Button	Zoomin	放大地图
Button	Zoomout	缩小地图
Button	Myaddr	返回到我的位置

2．Java代码

在 Java 部分中，我们依然首先进行整体的设计。

在整体部分中，我们完成了实时监听位置变化的功能。在获得一个新的位置时，通过自定义的 getAddrByGeoPoint()方法得到真实的地理名称，并显示在 TextView 中。

```
import ......     //省略部分导入
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import android.location.Address;
```

```
import android.location.Criteria;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;

public class MapDemo2 extends MapActivity {
    MapView mv;
    MapController controller;
    LocationManager manager;
    String locationProvider;
    public LocationListener locationListener;
    GeoPoint curPoint;
    Button b_zoomin;
    Button b_zoomout;
    Button b_myaddr;
    Button b_search;
    EditText et;
    TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        initView();
        //得到 LocationManager 对象
        manager = (LocationManager) getSystemService(Context.LOCATION_
        SERVICE);
        //得到提供商
        locationProvider = getLocationProvider(manager);
        //新建位置监听器
        locationListener = new LocationListener()
        {
         @Override
         public void onLocationChanged(Location location)
         {
             //从 Location 中获得 GeoPoint 对象
             curPoint = getGeoPoint(location);
             //将地图移动到该地点
             controller.animateTo(curPoint);
             String addr = getAddrByGeoPoint(curPoint);
             tv.setText(addr);
         }

         @Override
         public void onStatusChanged(String provider, int status, Bundle
         extras)
         {

         }

         @Override
         public void onProviderEnabled(String provider)
         {

         }

         @Override
         public void onProviderDisabled(String provider)
```

```
        {

        }
       };
       //开始监听位置变化信息
       manager.requestLocationUpdates(locationProvider, 1000, 10,
       locationListener);
    }


    @Override
    protected boolean isRouteDisplayed()
    {
        return false;
    }
}
```

接下来，我们需要实现的是 initView()方法，在该方法中我们要获得 xml 界面中所有组件的操作对象，并为每个按钮设置监听事件。

实现 initView()方法：

```
public void initView()
    {
        mv = (MapView) findViewById(R.id.mv);
        b myaddr = (Button) findViewById(R.id.myaddr);
        b zoomin = (Button) findViewById(R.id.zoomin);
        b zoomout= (Button) findViewById(R.id.zoomout);
        b search = (Button) findViewById(R.id.search);
        et = (EditText) findViewById(R.id.et);
        tv = (TextView) findViewById(R.id.tv);
        tv.setBackgroundColor(Color.GRAY);
        //得到 MapView 的控制器
       controller = mv.getController();
       //设置地图缩放等级，参数值在 1～21 之间
       controller.setZoom(10);
         OnClickListener l = new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                int id = v.getId();
                switch(id)
                {
                case R.id.zoomin:
                {
                    zoomIn();
                    break;
                }
                case R.id.zoomout:
                {
                    zoomOut();
                    break;
                }
                case R.id.myaddr:
                {
                    if (curPoint != null)
                    {
                        controller.animateTo(curPoint);
                    }
                    else
```

```
                    {
                        Toast.makeText(getBaseContext(), "暂时还未获得您的位
                        置", Toast.LENGTH_SHORT).show();
                    }
                    break;
                }
                case R.id.search:
                {
                    //得到编辑框中的输入
                    String addrName =  et.getText().toString();
                    //通过自定义的方法得到经纬度
                    GeoPoint point = getGeoPointByAddr(addrName);
                    //将地图定位到该点
                    controller.animateTo(point);
                    break;
                }
                }
            }
        };
        //为这些按钮设置监听事件
        b_zoomin.setOnClickListener(l);
        b_zoomout.setOnClickListener(l);
        b_myaddr.setOnClickListener(l);
        b_search.setOnClickListener(l);
    }
```

最后，我们还需要实现本例中最重要的两个方法：getAddrByGeoPoint()以及 getGeoPointByAddr()。

实现 getAddrByGeoPoint()方法：

```
public String getAddrByGeoPoint(GeoPoint point)
    {
        String addr = "";
        try
        {
            //新建解码器
            Geocoder coder = new Geocoder(this,Locale.getDefault());
            //得到经纬度
            double latitude = point.getLatitudeE6()/1E6;
            double longitude = point.getLongitudeE6()/1E6;
            //得到地址
            List<Address> list = coder.getFromLocation(latitude, longitude, 1);
            StringBuilder builder = new StringBuilder();
            //判断查询是否有结果
            if (list.size() > 0)
            {
                Address address = list.get(0);
                //得到最大的行数，之后循环读取其中的内容
                for(int i = 0; i < address.getMaxAddressLineIndex();i++)
                {
                    builder.append(address.getAddressLine(i)+"\n");
                }
                //得到邮编
                builder.append("邮编: "+address.getPostalCode()+"\n");
                //得到国家
                builder.append("国家: "+address.getCountryName());
                addr = builder.toString();
                Log.i("TAG", addr);
```

```
            }
            else
            {
                Toast.makeText(getBaseContext(), "无法解析到地名",
                Toast.LENGTH_SHORT).show();
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        return addr;
    }
```

这里需要注意的是，得到经纬度后还需要除以 1E6，才是真正的实际经度值和纬度值，才可以执行查询。实现 getGeoPointByAddr()方法：

```
public GeoPoint getGeoPointByAddr(String addr)
    {
        GeoPoint point = null;
        try
        {
            //新建解码器
            Geocoder coder = new Geocoder(this,Locale.getDefault());
            //得到地址
            List<Address> list = coder.getFromLocationName(addr, 1);
            if (list.size() > 0)
            {
                Address address = list.get(0);
                //得到经纬度
                double latitude = address.getLatitude()*1E6;
                double longitude = address.getLongitude()*1E6;
                //得到 GeoPoint 对象
                point = new GeoPoint((int)latitude,(int)longitude);
            }
            else
            {
                Toast.makeText(getBaseContext(), "无法解析地名", Toast.
                LENGTH SHORT).show();
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        return point;
    }
```

需要注意的是，这里得到的经纬度值又必须乘以 1E6 之后才可以在我们的 GoogleMap 上正确显示。由于这些操作都是经过网络传输的，在执行的过程中无法避免地会出现各种异常，所以这些操作我们最好在一个 try{…}catch(){…}语句块中执行。

到这里，我们的实例就完成了。接下来运行一下程序吧，效果如图 12.12 所示。按下 search 按钮后，地图显示到苏州的虎丘地区，如图 12.13 所示。

下一节我们将讲解如何使用 Overlay 在地图上显示文字内容。使用 Overlay 可以在特定的地理坐标点旁边进行注释性地说明，使用起来非常友好。

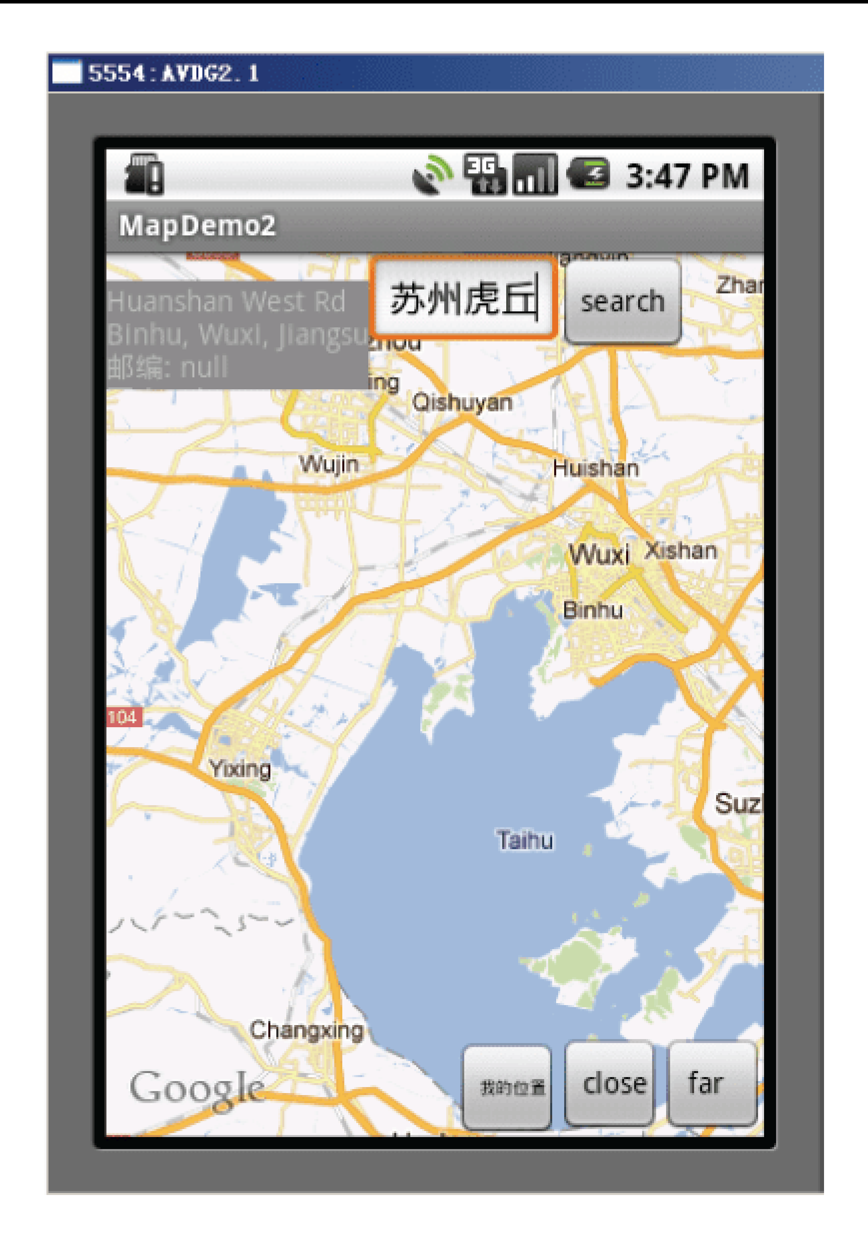


图 12.12　左上角显示现在的地理位置信息



图 12.13　单击查询按钮后，地图定位到虎丘

12.4　使用 Overlay

在地图中如果要显示一些注释内容，如提示用户现在所在的位置、显示现在的具体信息等。如果继续使用 TextView 无疑是不合适的，这会显得我们的应用非常简陋，就只是一些组件拼凑起来的。这个时候使用 Overlay 就可以产生比较好的效果了。Overlay 顾名思义，就是涂层的意思，它可以在 MapView 中指定的地方产生一小片涂层，用以显示具体的信息。

12.4.1　实现 Overlay 类

在程序中要使用 Overlay 时，首先要实现一个继承自 Overlay 的自定义类，并重写其 draw()方法。接下来，让我们开始实现自己的 Overlay 类吧，步骤如下：

（1）实现位置投影，将 GeoPoint 对象转换为屏幕上的像素点。

（2）创建图形对象和颜色对象。

（3）在画布上将图形画出来，如果要显示文字则直接使用 drawText()方法画在画布上。

这里的一些概念大家可能还比较陌生，现在看来似乎实现该类非常麻烦，但只要跟随笔者的脚步，聪明的读者，你会发现其实这也不是很“高难度”。

1．实现位置投影

投影——Projection 的作用是将地理位置坐标点转换为在屏幕上的像素坐标点。再简单一点说，就是将地图上的经纬度信息转换为屏幕上的 x/y 轴坐标。

实现投影类时，需要 4 个步骤：

（1）得到投影对象

得到投影对象的方法并不是使用其构造方法，而是从 MapView 对象中得到，方法

如下：

```
MapView.getProjection()
```

（2）新建一个地理位置点

该点就是你希望在地图上显示的点了，依然使用构造方法：

```
GeoPoint.GeoPoint(int latitudeE6, int longitudeE6)
```

（3）新建一个像素点类

像素点对象就是 android.graphics.Point 类对象，新建时，我们使用其无参构造方法：

```
Point.Point()
```

（4）将地理位置点投影到像素点位置

这是转换的最后一步，之前的 3 步都还只是“热身”，这一步的方法才是实现投影的关键，方法如下：

```
Projection.toPixels(GeoPoint in, Point out)
```

很显然，这里的第一个参数是要被转换的点，第二个参数是转换成功之后的点。

2. 创建图形和颜色对象

这里的图形对象是 android.graphics.RectF 类，该类的作用是使用指定的坐标实现一个矩形。说是矩形，实际上我们可以将之假设为一个未知的图形，矩形只是默认的图形而已。其构造方法如下：

```
RectF.RectF(float left, float top, float right, float bottom)
```

该方法的 4 个参数分别表示矩形的左边、上边、右边和底边。在新建时要确认左边的值要小于右边的值，上边的值要小于底边的值。Android 中的坐标系如图 12.14 所示。

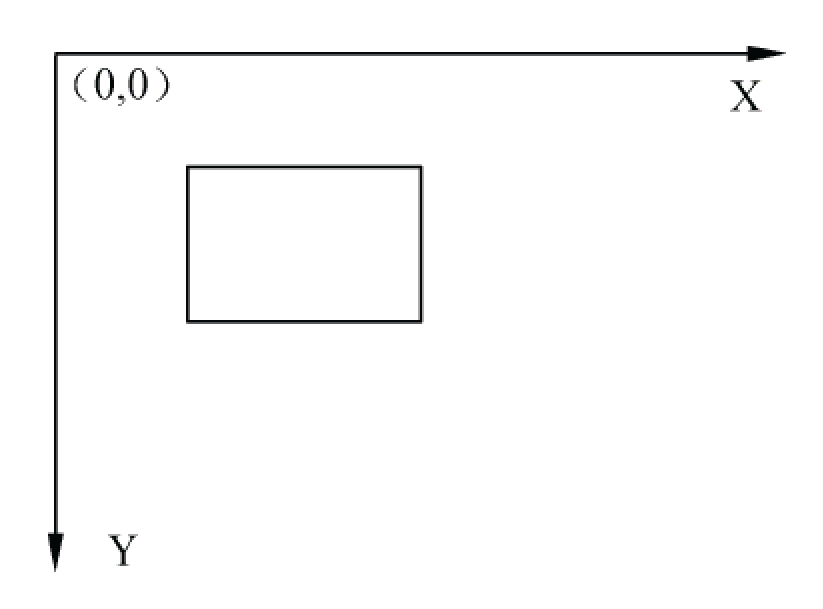


图 12.14　像素坐标系

颜色对象是 android.graphics.Paint 类对象，它的作用是实现图像的着色。新建时同样使用无参构造方法：

```
Paint.Paint()
```

接着，我们通常使用如下方法分别设置颜色和透明度：

```
Paint.setColor(int color)              //设置颜色
Paint.setAlpha(int a)                  //设置透明度
```

3．在画布上将图形画出来

画布对象是类 android.graphics.Canvas 的对象。Canvas 对象我们可以将之形象地理解为一个画布，我们可以在上面画出图像。在 draw()方法中，会传递有 canvas 对象，我们可以直接使用它进行绘画，常用方法如表 12-7 所示。

表 12-7　Canvas常用方法

方　法　名	作　　用
drawBitmap(Bitmap bitmap, float left, float top, Paint paint)	画出一个指定的图像
drawLine(float startX, float startY, float stopX, float stopY, Paint paint)	画出一条直线
drawCircle(float cx, float cy, float radius, Paint paint)	画出一个圆
drawOval(RectF oval, Paint paint)	画出一个椭圆
drawPoint(float x, float y, Paint paint)	画出一个点
drawRect(RectF rect, Paint paint)	画出一个矩形
drawRoundRect(RectF rect, float rx, float ry, Paint paint)	画出一个圆角矩形，常被用于背景
drawText(String text, float x, float y, Paint paint)	画出要输出的文字

12.4.2　通过实例学习 Overlay

通过上一小节的讲解，我们已经了解了使用 Overlay 需要掌握的基础知识。接下来，我们就尝试着将这些知识使用在真正的开发中。本例在 12.3.2 小节的实例上进行进一步的开发，增加的功能是：

（1）使用 Overlay 在 GoogleMap 上我的当前坐标点上画一个红点。

（2）使用 Overlay 在坐标点旁边添加注释性说明文字“我的位置”。

（3）在 MapView 中添加 View，显示具体的信息。

以上 3 个功能可以说是地图开发中非常实用也非常重要的功能，读者可以通过本例实现一个自己的 Overlay，这样以后就可以避免重复的劳动了。

1．实现MyOverlay

实现 MyOverlay 时，首先继承 Overlay，接着添加其构造方法，最后重写其中的 draw()方法。具体的代码如下所示：

```
import ……                                   //省略部分导入
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.RectF;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.Projection;

public class MyOverlay extends Overlay
{
    long latitude;                           //经度信息
    long longitude;                          //纬度信息
```

```
    String content;                  //要显示的内容

    public MyOverlay(long latitude,long longitude,String content)
    {
        this.latitude = latitude;
        this.longitude  =  longitude;
        this.content = content;
    }

    public MyOverlay(GeoPoint point,String content)
    {
        this.latitude   = point.getLatitudeE6();
        this.longitude  =  point.getLongitudeE6();
        this.content = content;
    }

    @Override
    public boolean draw(Canvas canvas, MapView mapview, boolean shadow, long when)
    {
        //新建投影数据对象
        Projection projection = mapview.getProjection();
        //新建 GeoPoint 对象
        GeoPoint geoPoint = new GeoPoint((int)latitude,(int)longitude);
        //新建 Point 对象
        Point point = new Point();
        //将 GeoPoint 对象投影成可显示的 Point 对象
        projection.toPixels(geoPoint,point);
        //创建一个 RectF 对象
        RectF rect = new RectF(point.x-5,point.y-5,point.x +5,point.y+5);
        //新建颜料对象
        Paint paint = new Paint();
        //设置颜色为红色
        paint.setColor(Color.RED);
        //在画布上将椭圆画出来
        canvas.drawOval(rect,paint);

        //创建背景
        RectF backRect = new RectF(point.x+7,point.y-15,point.x +60,point.y+5);
        //新建背景的颜料对象
        Paint backPaint = new Paint();
        //设置字体颜色为白色
        backPaint.setColor(Color.GRAY);
        //设置透明度
        backPaint.setAlpha(100);
        //在画布上画出一个圆角矩形
        canvas.drawRoundRect(backRect, 5, 5, backPaint);

        //新建字体的颜料对象
        Paint textPaint = new Paint();
        //设置字体颜色为白色
        backPaint.setColor(Color.WHITE);
        //将内容显示在画布上，坐标点要在背景之间
        canvas.drawText(content, point.x+10, point.y, textPaint);
        return super.draw(canvas, mapview, shadow, when);
    }
}
```

这里我们新建了两个构造方法，不管是哪个方法，其最终的操作都是获得需要绘制的经度信息、纬度信息以及要显示的内容。

接着让我们在 Activity 中实现 Overlay 的添加，一共需要 3 步：

（1）得到 MyOverlay 对象。

（2）得到地图中已有的 Overlay 列表。

（3）将 MyOverlay 对象添加到队列中，方法如下：

```
//使用 Overlay 显示
            MyOverlay overlay = new MyOverlay(curPoint,"我的位置");
            //得到已有的 Overlay 列表
            List<Overlay> overlays = mv.getOverlays();
            //将新建的 Overlay 加入到列表中显示
            overlays.add(overlay);
```

接着我们要实现的是在 MapView 中显示自己的 View 对象。使用地图时经常会弹出一个小气泡类型的提示，看起来非常的友好。接着我们就在本例中将之实现。

首先，准备一张气泡的图片，如图 12.15 所示。

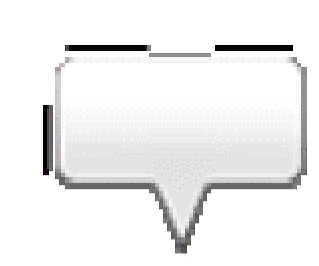


图 12.15　Pop.png

接着，在布局文件夹中新建一个布局，我们暂时将之命名为 pop.xml，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:background="@drawable/pop"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingLeft="5px"
    android:paddingTop="5px"
    android:paddingRight="5px"
    android:paddingBottom="20px"
>
   <TextView android:id="@+id/content"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:gravity="center_horizontal"
      android:textSize="10sp"
      android:textColor="#000"
      />
</LinearLayout>
```

代码非常地简单，这里的背景图就设置为了气泡。注意要设置 padding 属性，否则文字会与边界重合。

接着在 Java 代码部分实例化该布局，并得到其中的 TextView 操作对象，代码如下：

```
//实例化布局
LayoutInflater inflater = LayoutInflater.from(this);
        View  popView = inflater.inflate(R.layout.pop, null);
        popView.setOnClickListener(new OnClickListener()
        {
```

```
                @Override
                public void onClick(View v)
                {
                    v.setVisibility(View.GONE);
                }
            });
//得到 View 中的 TextView 对象
        pop_content = (TextView) popView.findViewById(R.id.content);
```

接下来的部分就是真正的添加部分了。首先为 popView 对象设置参数，在参数中包含宽、高、需要在地图上绘制的位置、View 的对齐位置等信息。方法为：

```
popView.setLayoutParams(new
MapView.LayoutParams(MapView.LayoutParams.WRAP CONTENT,
MapView.LayoutParams.WRAP CONTENT,curPoint,
                                    //GeoPoint 对象，也就是要添加的点坐标
 MapView.LayoutParams.BOTTOM_CENTER));
                                    //设置为底部中央对齐，也就是小气泡尖的位置
```

接着，我们可以在 popView 的 TextView 对象中添加要显示的内容：

```
pop_content.setText(addr);
```

最后，将 popView 添加到地图中：

```
mv.addView(popView);
```

这样，我们的程序就完成了！相比于 12.3.2 小节中的例子，我们只是修改了两个函数，分别是 initView()方法和 LocationListener 中的 onLocationChanged()方法。修改后的代码如下：

（1）initView()方法

```
public void initView()
    {
        //实例化 pop 布局
        LayoutInflater inflater = LayoutInflater.from(this);
        popView = inflater.inflate(R.layout.pop, null);
        popView.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                v.setVisibility(View.GONE);
            }
        });
        pop content = (TextView) popView.findViewById(R.id.content);
        //实例化 MapView
        mv       = (MapView) findViewById(R.id.mv);
        b myaddr = (Button) findViewById(R.id.myaddr);
        b zoomin = (Button) findViewById(R.id.zoomin);
        b zoomout= (Button) findViewById(R.id.zoomout);
        b search = (Button) findViewById(R.id.search);
        et = (EditText) findViewById(R.id.et);
        tv = (TextView) findViewById(R.id.tv);
```

```
......                    //此处省略按钮监听事件的实现，详见 12.3.2 小节中的代码
}
```

（2）修改之后的 LocationListener

```
locationListener = new LocationListener()
       {
        @Override
        public void onLocationChanged(Location location)
        {
            //从 Location 中获得 GeoPoint 对象
            curPoint = getGeoPoint(location);
            //将地图移动到该地点
            controller.animateTo(curPoint);
            //得到真实地址
            String addr = getAddrByGeoPoint(curPoint);
            //设置 popView 的属性
            popView.setLayoutParams(new MapView.LayoutParams(MapView.
            LayoutParams.WRAP_CONTENT, MapView.LayoutParams.WRAP_CONTENT,
            curPoint, MapView.LayoutParams.BOTTOM_CENTER));
           //在气泡中要显示的内容
           pop_content.setText(addr);
           //将 popView 添加到地图中
           mv.addView(popView);
            //使用 Overlay 显示
            MyOverlay overlay = new MyOverlay(curPoint,"我的位置");
            //得到已有的 Overlay 列表
            List<Overlay> overlays = mv.getOverlays();
            //将新建的 Overlay 加入到列表中显示
            overlays.add(overlay);
        }

        @Override
        public void onStatusChanged(String provider, int status, Bundle extras)
        {
            //状态改变时
        }

        @Override
        public void onProviderEnabled(String provider)
        {
            //服务可用时
        }

        @Override
        public void onProviderDisabled(String provider)
        {
            //服务不可用时
}
     }
```

运行以上代码，得到效果如图 12.16 所示。



图 12.16　使用 Overlay

12.5　小　　结

本章讲解了 Android 中地图开发的相关知识，包括 GoogleMap API 金钥的申请、Google 地图的显示、GPS 信息的获取以及 Overlay 的使用。本章重点是 GPS 的使用和地理位置编码的使用，包括从经纬度信息转换为地址名，以及其反转。难点是 Overlay 的使用，这还需要读者在以后的学习中进一步研究和使用。下一章我们将开始把之前所有的内容整合起来进行创意小应用的开发。

第 4 篇　项目案例开发

第 13 章　联系人助手

本章将动手完成一个创意小工具——联系人助手。该软件的主要功能是将 Excel 联系人表格导入到手机的联系簿中，或者将手机中的联系人导出到 Excel 中备份。实现该软件的过程中可以巩固如下几个方面的知识：

（1）多种布局和嵌套布局。

（2）Activity 的使用以及生命周期的管理。

（3）ContentProvider 的使用。

（4）ListView 以及 Adapter 的结合。

（5）文件存储。

（6）JXL 库的使用。

13.1　Jxl 简介

Jxl.jar 是一个通过 Java 操作 Excel 的工具类库，使用它我们可以很方便地操作 Excel 文件，它支持字体、数字、日期操作，支持图像和图表，基本可以满足我们的大部分需求，对中文的支持也比较好。最关键的是这套 API 是纯 Java 的，并不依赖 Windows 系统，即使运行在 Linux 下，它同样能够正确地处理 Excel 文件。

13.1.1　使用导入 jxl.jar

如果在工程中需要使用 jxl 的 API，首先先到网上下载 jxl.jar 包，接着我们还必须将其导入到目标工程中。导入的步骤是：

（1）选中目标工程，在右键菜单中选择 Build Path，接着在子菜单中选中 Configer Build Path，如图 13.1 所示。

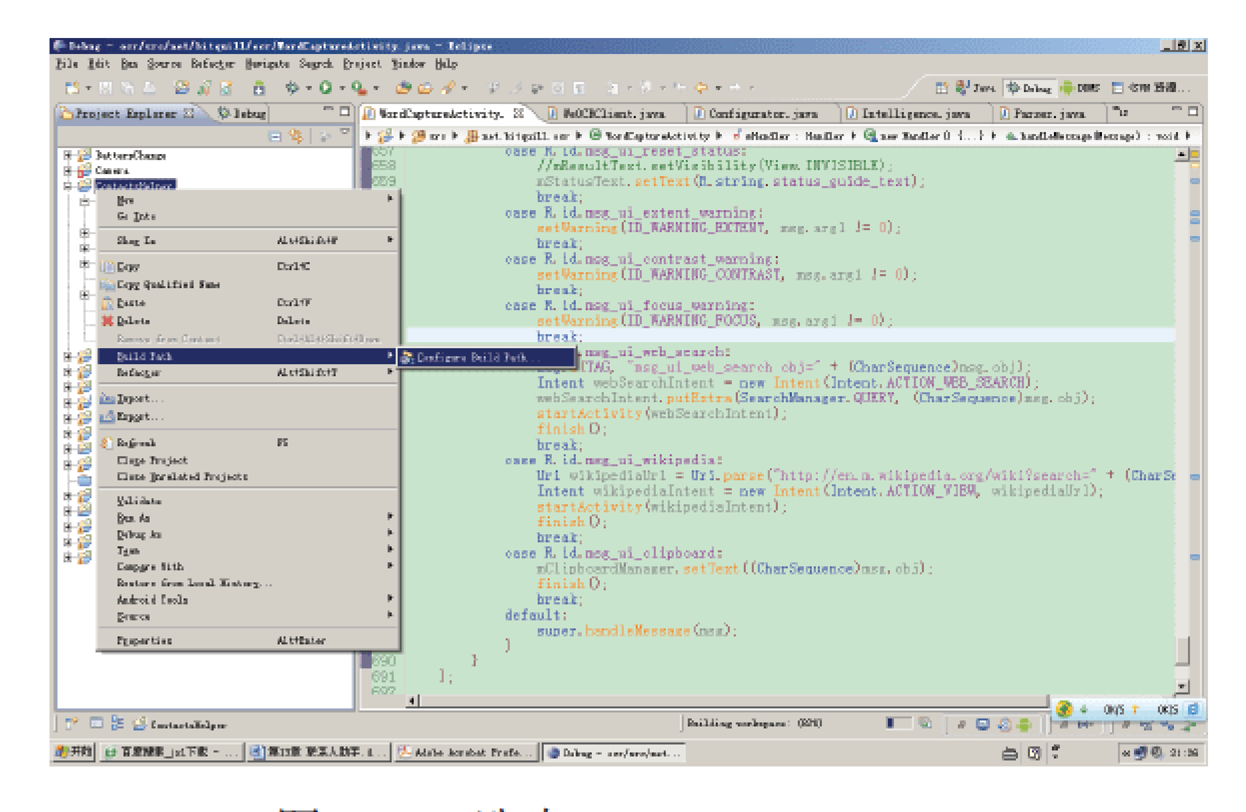


图 13.1　选中 Configer Build Path

（2）在弹出对话框中的左侧面板中选中 JavaBuildPath 选项，并在右侧面板中选中 Libraries 选项卡，如图 13.2 所示。

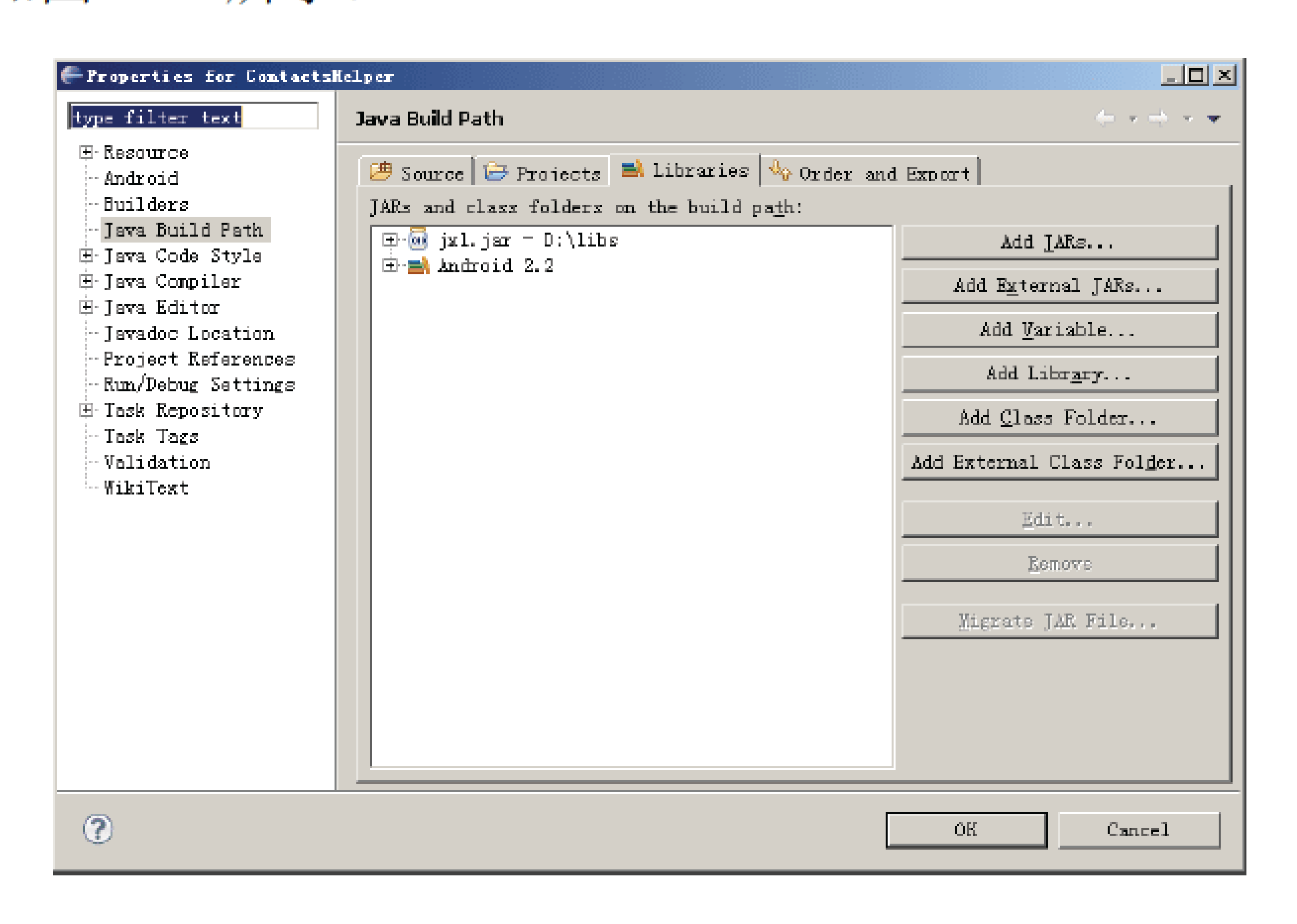


图 13.2　选中 Libraries

（3）单击 Add External Jars 按钮，弹出对话框如图 13.3 所示。

（4）在 JAR Selection 对话框中选择要添加的 Jar 包，单击“打开”按钮，并确定。

（5）浏览工程结构目录，此时发现 jxl.jar 已经成功添加到工程中了，此时工程目录结构如图 13.4 所示。

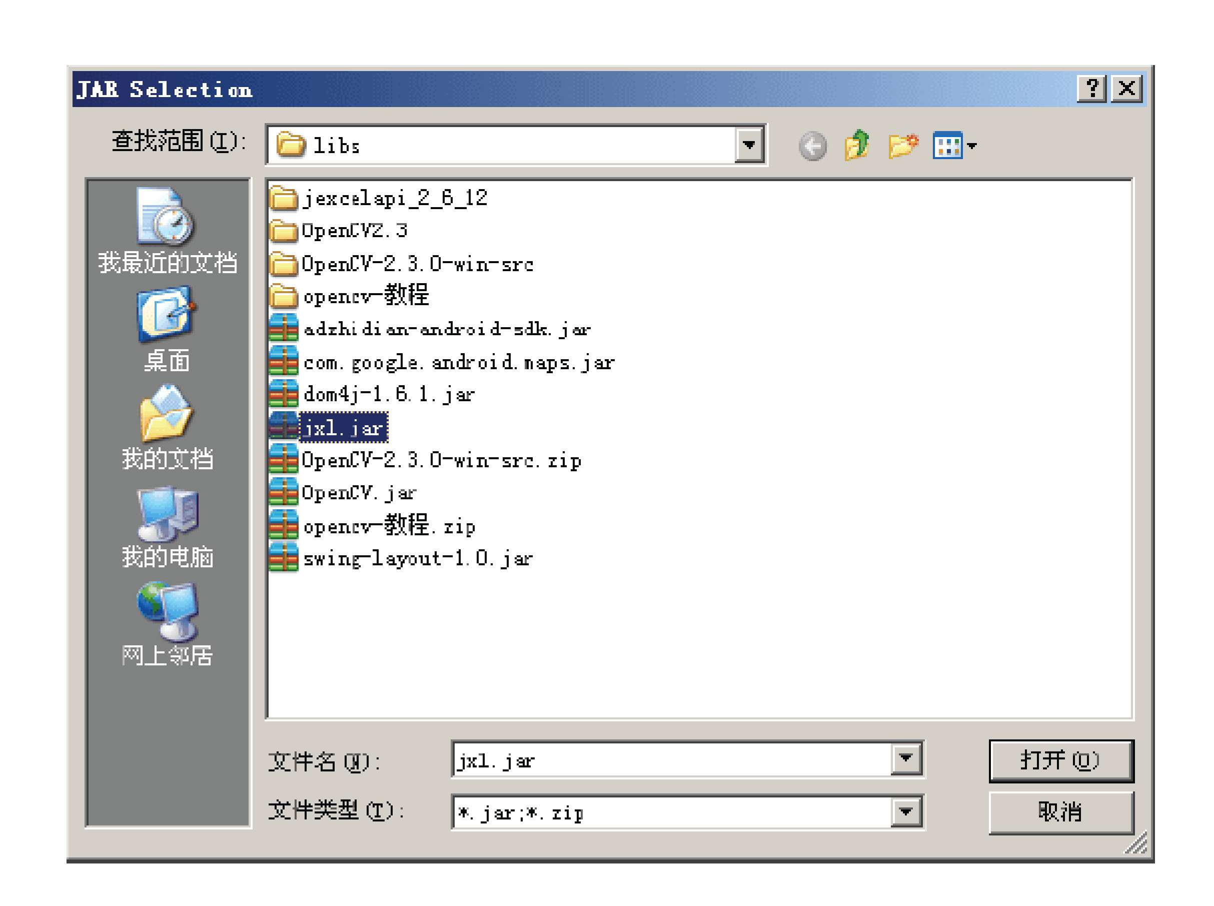


图 13.3　选中要添加的 Jar 包

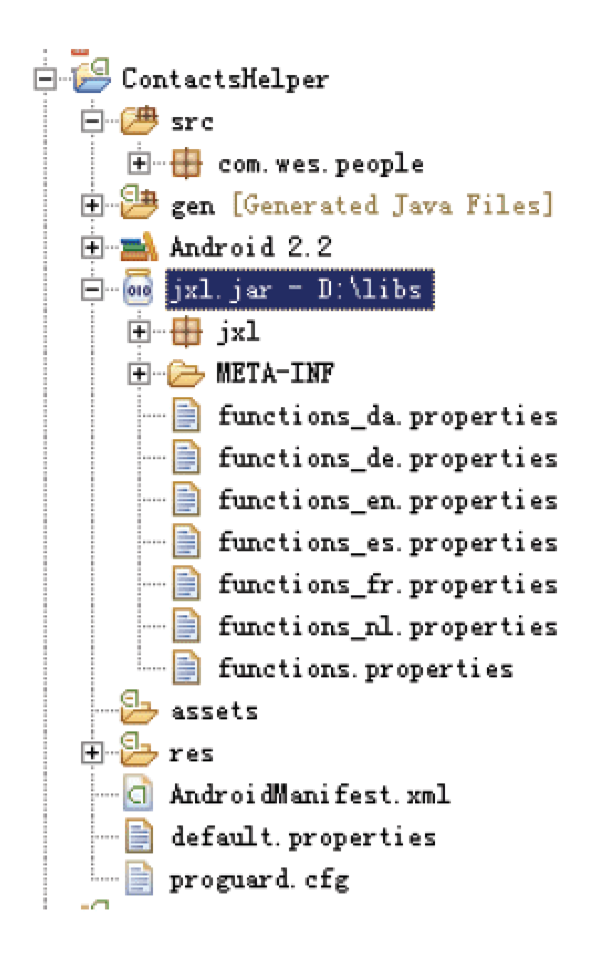


图 13.4　添加 jxl.jar 后的工程结构

（6）在程序中声明 import，大致如下所示：

```
import jxl.Sheet;
import jxl.Workbook;
import jxl.read.biff.BiffException;
```

13.1.2　使用 jxl 读取 Excel 文件

读取 Excel 时需要如下几个步骤：

（1）从文件中获得工作簿——Workbook。

（2）从 Workbook 中得到要操作的表——sheet。

（3）从表中读取单元格——Cell。

（4）从单元格中读取数据。

接下来，我们进行以上 4 个步骤的详细讲解。

1. 获得Workbook

获得 Workbook 时，我们不使用其构造函数，而是通过如下方法：

```
Workbook.getWorkbook(File file) throws IOException, BiffException
```

这里的参数就是我们需要操作的 Excel 文件了。

2. 获得表

我们知道一个 Excel 文件中可能存在好几张表，所以我们还需要得到具体要操作的表。同样地，我们通过 get()方法得到表的操作对象，方法如下：

```
Workbook.getSheet(int arg0) throws IndexOutOfBoundsException
```

3. 读取单元格

读取单元格时，需要行和列两个参数，语法格式如下：

```
Sheet.getCell(int col, int row)
```

需要注意的是，这里的第一个参数是列数，第二个参数是行数，读者使用时千万不要混淆了。

4. 从单元格中读取数据

从单元格中读取数据时，并没有严格的类型规定，我们可以将之一律视为字符串类型，方法为：

```
Cell.getContents()
```

方法返回的值是 String 类型，使用非常方便。以下是一个读取 Excel 的范例方法：

```
public boolean readXls()
    {
        String path = CONTACTS_RES;                //文件路径
    try
        {
        File file = new File(path);
        String fileName = file.getName();
        if (fileName.endsWith(".xls"))
        {
            Workbook  book  =  Workbook.getWorkbook( file); //得到工作簿
            Sheet sheet  =  book.getSheet( 0 );      //获得第一个工作表对象
            String name = sheet.getCell(0, 0).getContents();
                                                    //得到第一行第一列的数据
            String number = sheet.getCell(1, 0).getContents();
                                                    //得到第一行第二列的数据
            return true;
        }
        }
        catch (BiffException e)
```

```
            {
                e.printStackTrace();
            }
            catch (FileNotFoundException e)
            {
                e.printStackTrace();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
          return false;
      }
```

13.1.3　使用 jxl 创建 Excel 文件

使用 jxl 创建文件时同样需要 4 个步骤：

（1）创建可写的工作簿。

（2）创建可写的表。

（3）创建单元格。

（4）将单元格添加到表中。

接下来继续进行 4 个步骤的详细解析，首先观察第一步。

1．创建可写的工作簿

类似地，我们依然不使用构造方法进行工作簿与表的创建，语句如下：

```
Workbook.createWorkbook(File file) throws IOException
```

2．创建可写的表

得到了可写的工作簿之后，我们还需得到可写的表，方法如下：

```
WritableWorkbook.createSheet(String name, int index)
```

这里有两个参数，第一个参数是该表的名字；第二个参数是该表的索引，如果是第一张表则设置 index 为 0。

3．创建单元格

创建单元格时，我们就需要使用其构造方法了：

```
Label.Label(int c, int r, String cont)
```

这里的 3 个参数分别表示：

- c：该单元格的列数。
- r：该单元格的行数。
- cont：该单元格的内容。

4．将单元格添加到表中

最后使用 add()方法将单元格添加到表中，方法为：

```
WritableSheet.addCell(WritableCell cell) throws WriteException, Rows-
ExceededException
```

因为单元格 cell 中已经包含了所需要使用的信息，所以此时我们不需再添加额外的参数了。

最后，依然给出一个使用 jxl.jar 创建 Excel 文件的范例代码：

```
public boolean createXls()
    {
        String path = FILE_PATH + "/" + FILE_NAME.trim();      //文件路径
        try
        {
            File file = new File(path);
            if (!file.exists())
            {
                file.createNewFile();
            }
            WriteableWorkbook wwb = Workbook.createWorkbook(file);
                                                       //得到可写的工作簿
            WriteableSheet    ws = wwb.createSheet("联系人表", 0);
                                                       //得到可写的表
            Label labelC0 = new Label(0, 0, "姓名");
                                                 //创建第一行第一列的单元格
            Label labelC1 = new Label(1, 0, "号码");
                                                 //创建第一行第二列的单元格
            ws.addCell(labelC0);                 //添加单元格 1
            ws.addCell(labelC1);                 //添加单元格 2
            return true;
            }
            catch (FileNotFoundException e)
            {
                e.printStackTrace();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
            return false;
        }
}
```

13.2　界 面 规 划

关于界面，我们设计了登录时出现一个主界面，在主界面上可以选择导出联系人、导入联系人等操作。所以最简单的界面设计只需 3 个 Activity 就可以了。实际上，我们还需要设计一个文件浏览页面，以方便用户选择要导入的文件，以及存放文件的路径。

13.2.1　主界面实现

在规划中，主界面主要显示 3 个按钮，分别是：

（1）导出联系人。

（2）导入联系人。

（3）退出本应用。

为了使界面更加美观，我们需要将这 3 个按钮放在屏幕的中心位置，实现方法（Login.xml）如下所示：

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@drawable/default bg"
    android:orientation="vertical"
    android:layout height="fill parent"
   android:layout_width="wrap_content"
>
<LinearLayout
  android:layout_height="wrap_content"
   android:layout_width="200dp"
     android:orientation="vertical"
   android:background="@drawable/login_bg"
    android:layout_marginTop="60dp"
    android:layout_gravity="center">
    <TextView
    android:layout_width="fill_parent"
        android:text="联系人助手"
        android:layout height="60dp"
        android:textSize="30sp"
         android:layout marginTop="15dp"
         android:layout gravity="center"
         android:textColor="#000"
    />
         <Button
        android:layout_width="fill_parent"
        android:text="导出联系人"
        android:layout height="60dp"
        android:id="@+id/button1"
        android:textSize="20sp"
         android:layout marginTop="15dp"
        >
        </Button>

        <Button android:layout width="fill parent"
        android:text="导入联系人"
        android:layout_height="60dp"
        android:layout_gravity="fill"
        android:id="@+id/button2"
        android:textSize="20sp"
        android:layout marginTop="15dp"
         >
        </Button>

        <Button android:layout width="fill parent"
        android:text="退出本应用"
        android:layout_height="60dp"
        android:layout_gravity="fill"
        android:id="@+id/button3"
        android:textSize="20sp"
        android:layout_marginTop="15dp"
        android:layout_marginBottom="20dp"
         >
```

```
        </Button>
    </LinearLayout>
</LinearLayout>
```

以上代码我们使用了两层 LinearLayout 嵌套，此时界面显示如图 13.5 所示。



图 13.5　主界面

13.2.2　导出文件、导入文件界面的实现

当用户单击“导出联系人”按钮后，进入导出文件界面。该界面中我们需要提示用户选择文件的保存位置，并提示用户需要保存为的文件名，同时还需要添加一个进度条，用以显示程序完成的进度等。

实现代码（Export.xml）如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/default_bg"
>
<!--关系布局，用以提示用户输入文件名 -->
    <RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="50dp"
    >
    <TextView
    android:id="@+id/name"
    android:layout_width="wrap_content"
    android:layout_height="50dp"
    android:text="文件名："
    android:gravity="center_vertical"
    android:layout_alignParentLeft="true"
    android:textSize="20sp"
    />
    <EditText
    android:id="@+id/nameEdit"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
    android:text=""
    android:layout toRightOf="@+id/name"
    android:textSize="20sp"
    />
    </RelativeLayout>
    <TextView
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:text="选择文件保存位置"
    android:gravity="center_vertical"
    android:textSize="20sp"
    android:paddingTop="5dp"
    />

    <TextView
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:text="(路径为空则直接在保存/sdcard 下): "
    android:gravity="center vertical"
    android:textSize="18sp"
/>
<!--关系布局，用以提示用户输入文件保存位置，或单击进入文件浏览界面 -->
    <RelativeLayout
    android:layout width="fill parent"
    android:layout height="80dp"
    >
    <EditText
    android:id="@+id/filePath"
    android:layout width="fill parent"
    android:layout height="80dp"
    android:text=" "
    android:paddingLeft="10px"
    android:layout toLeftOf="@+id/search"
    />
    <Button
    android:id="@+id/search"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="浏览"
    android:layout alignParentRight="true"
    />
    </RelativeLayout>
    <Button android:layout height="wrap content"
    android:layout width="wrap content"
    android:id="@+id/btn"
    android:text="开始导出! "
    android:layout_gravity="center_horizontal"
    ></Button>
    <TextView
    android:id="@+id/tip"
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:text="正在导出:"
    android:gravity="center vertical"
    android:textSize="20sp"
    />
     <TextView
     android:id="@+id/tv"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
```

```
    android:textSize="20sp"
    android:paddingLeft="20px"
/>
<!--进度条，用以显示完成情况 -->
<ProgressBar
   android:id="@+id/bar"
    android:layout_width="300sp"
    android:layout_height="wrap_content"
    style="?android:attr/progressBarStyleHorizontal"
    android:paddingLeft="20px"
   />
</LinearLayout>
```

运行效果如图 13.6 所示。这里的界面使用了 LinearLayout 和 RelativeLayout 的嵌套。两个关系布局都作为了父线性布局的子布局。这样的布局嵌套还需要读者自己去摸索和熟练，笔者的布局只能是抛砖引玉了。同样的界面，我们也可以用来显示导入界面，只需稍作修改即可，这里就不再给出导入界面的代码，最后效果如图 13.7 所示。



图 13.6　导出界面



图 13.7　导入界面

最后，我们实现文件浏览界面，在文件浏览界面中，我们只需一个 ListView 就可以了：

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical"
>
  <ListView
    android:id="@android:id/list"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
  />
</LinearLayout>
```

13.3　功 能 实 现

本节我们将开始功能的实现，我们需要实现的功能共有 3 个：

（1）导出联系人列表至 Excel，该功能设计在 ExportExcel.Java 中实现。
（2）导入 Excel 文件至联系人中，该功能设计在 ImportExcel.Java 中完成。
（3）文件浏览功能，该功能在 FileSearch.java 中完成。

13.3.1　实现导出联系人

首先，我们实现导出联系人功能。主要步骤为：

（1）通过 ContentProvider 查询所有的联系人数据至 Cursor 中。

（2）将 Cursor 中的数据遍历并保存到 Map 中，因为 Cursor 存在的时间很短，我们无法保证在很短的时间内就可以完成所有数据的写入工作。

（3）遍历 Map 中的数据并将之写入到 Excel 中。

1．整体设计

首先，进行 ExportExcel.java 的整体设计，在整体设计中我们需要完成所有变量的声明：

```
import ......                                          //省略部分导入

import jxl.Workbook;
import jxl.write.Label;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import android.content.ContentResolver;
import android.os.Environment;
import android.provider.ContactsContract;

public class ExportExcel extends Activity {
   /** Called when the activity is first created. */
    public String FILE_PATH    = " ";               //文件路径
    public String FILE_NAME    = "";                //文件名
    public static final String END_TAG = "导出完毕！！，可以退出本应用~~";
    WritableWorkbook wwb;                           //可写的工作簿
    WritableSheet ws;                               //可写的表
    ContentResolver resolver;                       //内容处理者
    TextView tv;                                    //显示程序状态的 TextView
    TextView tip;                                   //提示用户的 TextView
    EditText et;                                    //用于输入文件保存路径
    EditText fileNameEdit;                          //用于输入文件名
    Button btn;                                     //开始导出按钮
    Button searchBtn;                               //浏览文件按钮
    ProgressBar bar;                                //进度条
    Handler handler;                                //handler 用于处理消息，改变 UI
    HashMap<String,String> map = new HashMap<String, String>();
                                                    //保存联系人数据
    HashMap<String,String> map_new = new HashMap<String, String>();
    //保存已经导出的联系人
    @Override
public void onCreate(Bundle savedInstanceState)
 {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.export);
       initView();                  //界面初始化
```

```
        handler = new Handler()
        {
         public void handleMessage(Message msg)
         {
             String content = (String) msg.obj;
             int added = msg.arg1;
             tv.setText(content);                          //显示正在导出的人名
             bar.setProgress(added);                       //设置进度
             if (content.equals(END TAG))                  //判断是否导出完毕
             {
                 bar.setVisibility(View.INVISIBLE);        //使进度条消失
                 tip.setVisibility(View.INVISIBLE);        //使提示文本框消失
                 tv.append("\n 本次导出记录，" + String.valueOf(added)+"条");
             }
         }
        };
}
}
```

这里我们新建了一个 Handler，那么什么是 Handler 呢？我们知道在 Android 中所有的界面更新操作都必须在主线程中进行，那么如果我们希望在不同的线程中更新 UI 呢？这时使用 Handler 就可以达到我们的目的了。在 Android 中 Handler 常被用来接收其他线程发送的消息，并适时地进行界面的更新。

在新建 Handler 时，会重写其 handleMessage()方法，在该方法中就可以进行 UI 界面的修改了。在其他的线程中使用 Handler.sendMessage()方法就会触发该回调方法。在 add()方法中我们会继续讲解 sendMessage()方法的使用。

2. 实现initView()

接着我们需要完成的是 initView()函数的实现。在 initView()中我们需要完成界面的初始化，关键是按钮单击事件的实现。

```
public void initView()
    {
         tv = (TextView) findViewById(R.id.tv);
        tip = (TextView) findViewById(R.id.tip);
        bar = (ProgressBar) findViewById(R.id.bar);
        fileNameEdit = (EditText) findViewById(R.id.nameEdit);
        et = (EditText) findViewById(R.id.filePath);
        btn = (Button) findViewById(R.id.btn);
        searchBtn = (Button) findViewById(R.id.search);
        tip.setVisibility(View.INVISIBLE);
        bar.setVisibility(View.INVISIBLE);
        btn.setOnClickListener(new OnClickListener()
         {
             @Override
             public void onClick(View arg0)
             {
                 map.clear();
                 map new.clear();
                 FILE_NAME = fileNameEdit.getText().toString();//得到文件名
                 FILE_PATH = et.getText().toString();    //得到文件保存路径
                 if (initXls()) //初始化 jxl，使之可用，如果成功则开始添加操作
                 {
                     bar.setMax(map.size());
```

```
                    tip.setVisibility(View.VISIBLE);
                    bar.setVisibility(View.VISIBLE);
                    Thread t = new Thread(new Runnable()
                    {
                        @Override
                        public void run()
                        {
                            add();//使用单独的线程完成将联系人添加到 Excel 的工作
                        }
                    });
                    t.start();                          //启动线程
                }
            }
        });

        searchBtn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                Intent i = new Intent(ExportExcel.this,FileSearch.class);
                startActivityForResult(i,0);    //跳转到文件浏览界面
            }
        });
    }
```

3. 实现initXls()

该方法中，实现了可写的工作簿和表的创建，同时完成了数据源的组成。也就是将联系人列表读入到 HashMap 中。关于 jxl.jar 的使用，在 13.1 节中已经有了具体的讲解，这里就不再赘述，读者可以结合 13.1 节进行理解。

```
public boolean initXls()
{
    if (FILE_PATH.trim().equals(""))
    {
        FILE_PATH = Environment.getExternalStorageDirectory().
        getAbsolutePath();
    }
    if (!FILE_NAME.endsWith(".xls") && !FILE_NAME.contains("."))
    {
        FILE_NAME = FILE_NAME + ".xls";
    }
    String path = FILE_PATH + "/" + FILE_NAME.trim();
    try
    {
        File file = new File(path);
        if (!file.exists())
        {
            file.createNewFile();
        }
        wwb = Workbook.createWorkbook(file);        //得到可写的工作簿
        ws = wwb.createSheet("联系人表", 0);         //得到可写的表
    //得到 ContentResolver 对象
       resolver = getContentResolver();
       Cursor cursor = resolver.query(ContactsContract.CommonDataKinds.
       Phone.CONTENT_URI, null,
                null, null, null);
       cursor.moveToFirst();
```

```
        while(!cursor.isAfterLast())                    //遍历 Cursor
        {
         String name = cursor.getString(cursor.getColumnIndex
         (ContactsContract.Contacts.DISPLAY_NAME));          //得到姓名
         String number= cursor.getString(cursor.getColumnIndex
         (ContactsContract.CommonDataKinds.Phone.NUMBER)); //得到号码
         map.put(name, number);
         cursor.moveToNext();
        }
         return true;
    }
    catch (FileNotFoundException e)
    {
        e.printStackTrace();
    }
    catch (IOException e)
    {
        Toast.makeText(getBaseContext(), "路径设置错误", Toast.
        LENGTH_SHORT).show();
        e.printStackTrace();
    }
   return false;
}
```

4．实现add()方法

该方法的功能是遍历数据源，并将其中的数据逐一读取并新建单元格写入表中。由于add()方法被执行在不同的线程中，所以我们不能直接在该方法中进行 UI 界面的修改。而解决的办法就是使用之前提到的 Handler.sendMessage()方法。

```
public void add()
{
    try
    {
        Iterator<Entry<String, String>> iter = map.entrySet().iterator();
        Label labelC0 = new Label(0, 0, "姓名");
        Label labelC1 = new Label(1, 0, "号码");
        ws.addCell(labelC0);
        ws.addCell(labelC1);
        int row = 1;
        while(iter != null && iter.hasNext())       //遍历 map
        {
            Entry<String, String> entry = iter.next();
            String name = entry.getKey();
            String number = entry.getValue();
            //判断是否重复导出
            if (map_new.containsKey(number))
            {
                continue;
            }
            else
            {
                map_new.put(number, "");
            }
            //将姓名和号码写入到单元格中并添加到表里，同时通知 handler 此时正在导
              出的内容
            if (name != "" && number != "")
            {
```

```
                    Label labelName = new Label(0, row, name);
                    Label labelNum  = new Label(1, row, number);
                    ws.addCell(labelName);
                    ws.addCell(labelNum);
                    Message msg =  new Message();
                    msg.obj = name+" : "+number;
                    msg.arg1 = row;
                    handler.sendMessage(msg);          //将消息发送到 handler 中
                    row ++;

                }
            }
        wwb.write();                                   //写入数据
        wwb.close();                                   //关闭工作簿

        Message msg =  new Message();                  //发送结束标志
         msg.obj = END TAG;
         msg.arg1 = row - 1;
         handler.sendMessage(msg);
    }
    catch  (Exception e)
    {
        e.printStackTrace();
    }
}
```

请读者朋友们注意这里的 Handler.sendMessage()方法的使用。首先我们要新建一个消息，可以通过构造方法获得：

```
Message msg = new Message()
```

接着我们可以向该 Message 对象中添加参数，该参数可以是任何类的对象。一般情况下，为了分辨该消息属于哪个线程，需要执行怎样的操作，我们会给它附上一个标签。例如，我们可以将该 Message 定义为：

```
Message.what = 0;
```

这样在 handleMessage(Message msg)时，我们可以通过 if (msg.what == 0)来判断该消息是不是我们正在等待的消息。

最后，组织好 Message 以后，我们还需要将消息发送出去，发送到 Handler 的消息循环中，以便 Handler 进行接受和处理。方法为：

```
Handler.sendMessage(Message);
```

5．实现onActivityResult()方法

因为之前启动文件浏览页面时使用的是 startActivityForResult()方法，所以在本页面中还需实现 onActivityResult()方法。

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);

    switch (resultCode)
    {
```

```
    case RESULT_OK:
        Bundle bundle = data.getExtras();
        String filePath = bundle.getString("filePath");  //得到文件保存路径
        et.setText(filePath);
        break;
    default:
        break;
    }
}
```

13.3.2　实现导入联系人功能

实现导入联系人时，步骤与导出大同小异，主要工作就是遍历 Excel 文件，将数据逐一写入到联系人数据库中。

关于导入联系人，其代码框架与导出极其相似。所以这里就不再给出其代码，这里重点讲解其中的 add()方法，实现 add()方法的代码如下所示：

```
import ......                              //省略部分导入
import jxl.Sheet;
import jxl.Workbook;
import android.content.ContentResolver;
import android.provider.ContactsContract.CommonDataKinds.Phone;
import android.provider.ContactsContract.CommonDataKinds.StructuredName;

    public void add()
    {
        try
       {
        //得到 ContentResolver 对象
          resolver = getContentResolver();
          Cursor cursor = resolver.query(ContactsContract.CommonDataKinds.
          Phone.CONTENT_URI, null,
                null, null, null);
          cursor.moveToFirst();
          while(!cursor.isAfterLast())
          {
            String name = cursor.getString(cursor.getColumnIndex(Contacts-
            Contract.Contacts.DISPLAY_NAME));
            map.put(name, "");          //得到已有的联系人，以防止重复
            cursor.moveToNext();
          }

          int rows = sheet.getRows();

          for (int i = 1; i < rows; i++)
          {
            String name = sheet.getCell(0, i).getContents();
            String number = sheet.getCell(1, i).getContents();
            if (map_new.containsKey(name))
            {
                continue;
            }
            else
            {
                map_new.put(name, "");
            }
```

```
            if (name != "" && number != "")
            {
                if (map.containsKey(name))
                {
                    continue;
                }
                //得到 ContentValues 对象
                    ContentValues values = new ContentValues();
                    //得到新记录的 rawContentUri
                   Uri rawContentUri = resolver.insert(RawContacts.
                   CONTENT_URI, values);
                   //通过解析得到新记录的 Id
                   long rawContentId = ContentUris.parseId(rawContentUri);
                   //插入名字
                   values.clear();
                   values.put(Data.RAW_CONTACT_ID, rawContentId);
                   values.put(Data.MIMETYPE, StructuredName.
                   CONTENT_ITEM_TYPE);
                   values.put(StructuredName.DISPLAY_NAME, name);
                   resolver.insert(ContactsContract.Data.CONTENT_URI,
                   values);

                   //插入号码
                   values.clear();
                   values.put(Data.RAW_CONTACT_ID, rawContentId);
                   values.put(Data.MIMETYPE, Phone.CONTENT_ITEM_TYPE);
                   values.put(Phone.NUMBER, number);
                   values.put(Phone.TYPE, Phone.TYPE_MAIN);
                   resolver.insert(Data.CONTENT_URI, values);

                Message msg =  new Message();
                msg.obj = name+" : "+number;
                msg.arg1 = i;
                handler.sendMessage(msg);
                added ++;
            }
          }
          book.close();
          Message msg =  new Message();
            msg.obj = END_TAG;
            handler.sendMessage(msg);
      }
      catch  (Exception e)
      {
          e.printStackTrace();
      }
  }
```

这里需要注意的是，在开始添加之前我们要得到目前手机中的联系人列表，以防止重复输入。接着，在输入时，我们还需要判断 Excel 文件中是不是有重复的联系人，如果没有才能继续添加，如果已经有了则进行下一次循环。在每次插入数据时需要将 ContentValues 对象清空，也就是调用 ContentValues.clear()方法。

13.3.3　实现文件浏览功能

该功能用于浏览手机中现有的文件，以方便用户选择要导入的文件或者方便用户选择

要导出文件的保存路径。同样地，让我们首先进行整体设计。

1. 整体设计

在整体设计中我们完成了界面的初始化，并实现了列表的单击事件以及长按事件。

```
import ......                              //省略部分导入
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemLongClickListener;
import android.widget.ListView;
import android.widget.Toast;
import java.util.ArrayList;
import java.util.List;

public class FileSearch extends ListActivity
{
      private List<String> items=null;
      private List<String> paths=null;
      private String rootPath="/";
      private ListView mList;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.file_list);

        mList = (ListView)findViewById(android.R.id.list);
        mList.setOnItemLongClickListener(new OnItemLongClickListener()
        {
            @Override
            public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
            int position, long arg3)
            {
                File file=new File(paths.get(position));
                 if (!file.canRead())
                 {
                     Toast.makeText(getBaseContext(), "权限不够！", Toast.
                     LENGTH_SHORT).show();
                     return true;
                 }
                returnFilePath(file);
                return false;
            }
        });
        getFileDir(rootPath);
        Toast.makeText(this, "长按文件夹选择路径", Toast.LENGTH_LONG).show();
    }

protected void onListItemClick(ListView l,View v,
          int position,long id)
    {
         File file=new File(paths.get(position));
         if (!file.canRead())
         {
             Toast.makeText(this, "权限不够！", Toast.LENGTH_SHORT).show();
             return;
         }
```

```
        if (file.isDirectory())
        {
          getFileDir(paths.get(position));
        }
        else
        {
          returnFilePath(file);
        }
    }
}
```

2. 实现getFileDir()方法

该方法用以显示目前的文件路径，并显示该路径下的文件列表。该方法的奥秘就在于：

（1）通过传递进来的文件路径得到文件对象。

（2）通过 File.listFiles()方法得到该方法下的子文件列表。

（3）将文件列表下的文件名和文件路径都取出来并分别保存到两个 List 中。

（4）使用新组建的 List 构造 adapter。

（5）设置适配器进行界面显示。

该方法最后的代码如下所示：

```
private void getFileDir(String filePath)
{
     setTitle(filePath);                    //显示当前路径
    items=new ArrayList<String>();
    paths=new ArrayList<String>();
    File f=new File(filePath);
    File[] files=f.listFiles();             //得到子文件列表

    if(!filePath.equals(rootPath))
    {
      items.add("backRoot");
      paths.add(rootPath);
      items.add("back");
      paths.add(f.getParent());
    }

    for(int i=0;i<files.length;i++)
    {
      File file=files[i];                   //遍历所有的子文件列表
      items.add(file.getName());            //得到文件名
      paths.add(file.getPath());            //得到文件路径
    }
    setListAdapter(new MyAdapter(this,items,paths));
                                            //修改 adapter，使之显示现有的文件列表
}
```

3. 实现returnFilePath()方法

该方法用于得到文件的路径，并将之设为该 Activity 的 result，这样发起端的 Activity 就可以得到文件路径了。

```
     private void returnFilePath(File file)
```

```
    {
        String filePath =  file.getPath();
        Intent i = getIntent();
        Bundle bundle = new Bundle();
        bundle.putString("filePath",filePath);
        i.putExtras(bundle);
        FileSearch.this.setResult(RESULT OK, i);
        finish();
    }
}
```

4．实现MyAdapter类

最后，我们还要实现的就是 Adapter 了。该类的作用是将文件名列表与视图有效地绑定在一起。在构造时，我们需要两个数据源，分别是文件名的 List 和文件路径的 List。实现代码如下：

```
mport ......                              //省略部分导入
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.view.LayoutInflater;

public class MyAdapter extends BaseAdapter
{
  private LayoutInflater mInflater;
  private Bitmap bm1;
  private Bitmap bm2;
  private Bitmap bm3;
  private Bitmap bm4;
  private List<String> items;
  private List<String> paths;

  public MyAdapter(Context context,List<String> item,List<String> path)
  {
    mInflater = LayoutInflater.from(context);
    items = item;
    paths = path;
    bm1 = BitmapFactory.decodeResource(context.getResources(),
                                       R.drawable.back01);
    bm2 = BitmapFactory.decodeResource(context.getResources(),
                                       R.drawable.back02);
    bm3 = BitmapFactory.decodeResource(context.getResources(),
                                       R.drawable.folder);
    bm4 = BitmapFactory.decodeResource(context.getResources(),
                                       R.drawable.doc);
  }

  @Override
  public int getCount()
  {
    return items.size();
  }

  @Override
  public Object getItem(int position)
  {
    return items.get(position);
  }

  @Override
```

```
  public long getItemId(int position)
  {
    return position;
  }

  @Override
  public View getView(int position,View convertView,ViewGroup parent)
  {
    ViewHolder holder;

    if(convertView == null)
    {
      convertView = mInflater.inflate(R.layout.file_row, null);
      holder = new ViewHolder();
      holder.text = (TextView) convertView.findViewById(R.id.text);
      holder.icon = (ImageView) convertView.findViewById(R.id.icon);
      convertView.setTag(holder);
    }
    else
    {
      holder = (ViewHolder) convertView.getTag();
    }

    File f=new File(paths.get(position).toString());
    if(items.get(position).toString().equals("backRoot"))
    {
      holder.text.setText("返回根目录");
      holder.icon.setImageBitmap(bm1);
    }
    else if(items.get(position).toString().equals("back"))
    {
      holder.text.setText("返回上一级");
      holder.icon.setImageBitmap(bm2);
    }
    else
    {
      holder.text.setText(f.getName());
      if(f.isDirectory())                    //判断该文件是否有子目录
      {
        holder.icon.setImageBitmap(bm3);
      }
      else
      {
        holder.icon.setImageBitmap(bm4);
      }
    }
    return convertView;
  }

  private class ViewHolder
  {
    TextView text;
    ImageView icon;
  }
}
```

13.3.4　实现主界面跳转功能

此时，基本功能已经完成，我们还缺少一个“大脑”来支配这些功能。而这个大脑就

是 MainActivity，该 Activity 的作用就是跳转到不同的页面，作为一个指挥调度的平台而存在。实现代码如下所示：

```
import ......            //导入略

public class MainActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);

        Button b1 = (Button) findViewById(R.id.button1);
        Button b2 = (Button) findViewById(R.id.button2);
        Button b3 = (Button) findViewById(R.id.button3);

        OnClickListener listener = new OnClickListener()
        {

            @Override
            public void onClick(View v)
            {
                int id = v.getId();
                switch (id)
                {
                case R.id.button1:
                {
                    Intent i = new Intent(MainActivity.this,ExportExcel.class);
                    startActivity(i);
                    break;
                }
                case R.id.button2:
                {
                    Intent i = new Intent(MainActivity.this,ImportExcel.class);
                    startActivity(i);
                    break;
                }
                case R.id.button3:
                {
                    finish();
                    break;
                }
                }
            }
        };

        b1.setOnClickListener(listener);
        b2.setOnClickListener(listener);
        b3.setOnClickListener(listener);
    }

    @Override
    protected void onDestroy()
    {
        super.onDestroy();
    }

}
```

13.3.5　修改注册表

最后我们需要修改一个注册表，向其中添加一些权限的认证、一些 Activity 的声明等。最后代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.wes.people"
     android:versionCode="2"
     android:versionName="2.0">
   <uses-sdk android:minSdkVersion="8" />

   <application android:icon="@drawable/helper"
   android:label="联系人助手"
   >
      <activity android:name=".MainActivity"
              android:label="@string/app name"
              android:screenOrientation="portrait">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
       <activity android:name="ImportExcel"
              android:label="@string/app name"
              android:screenOrientation="portrait">
      </activity>
      <activity android:name="ExportExcel"
              android:label="@string/app name"
              android:screenOrientation="portrait">
      </activity>
      <activity android:name="FileSearch"
              android:label="@string/app name"
              android:screenOrientation="portrait">
      </activity>
      <activity android:name="com.waps.OffersWebView"
            android:configChanges="keyboardHidden|orientation" />
       <meta-data android:value="2E8C8372625A676EC945176489D80486"
       android:name="com.view.AdView.pid" />
       <meta-data android:value="zhidian" android:name="com.view.
       AdView.channel" />

   </application>
  <uses-permission android:name="android.permission.READ CONTACTS" />
   <uses-permission android:name="android.permission.WRITE CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_
    STORAGE" />
</manifest>
```

运行以上程序，以导入联系人为例，我们先来关注文件浏览界面的效果，如图 13.8 所示。

接着选中 gdkj.xls 文件，此时程序会返回到导入界面。单击“开始导入！”按钮，效果如图 13.9 所示。导入完毕时，效果如图 13.10 所示。



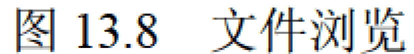

图 13.8　文件浏览



图 13.9　导入中界面

此时，再进入手机的 Contacts 应用中查看联系人，我们会发现联系人已经出现在界面中了，如图 13.11 所示。



图 13.10　导入完毕界面



图 13.11　查看联系人成功

13.5　小　　结

本章我们巩固了 Layout、ContentProvider、Activity、Intent 以及 ListView 等相关知识，同时学习了 Handler 的使用。本章的重点在于使用 Handler 在不同的线程中修改界面，难点是 jxl.jar 以及 ContentProvider 的使用。在下一章中，我们将动手编写一个功能比较强大的轨迹跟踪器，显然，这是一个 LBS 相关开发。

第 14 章　个人轨迹跟踪器

本章我们将编写一个个人的轨迹跟踪器，用它可以记录你外出旅游经过的各个地点，并在地图上画出轨迹，同时还可以查看经过的距离、平均的移动速度等。要实现该软件我们要用到以下几个方面的知识：

- 数据库存储操作。
- GoogleMap 的开发。
- Service 的使用。
- Activity 和 MapActivity 的使用。
- 一些核心的算法，如通过两个经纬度坐标点求出它们之间的距离。

14.1　界面 UI 实现

一般情况下实现一个软件的基本流程如下：

（1）设计各个 Activity 的界面并实现。

（2）实现数据库的设计和实现。

（3）实现程序的功能以及各个 Activity 的连接。

（4）测试各功能效果。

本节的工作就是完成第一步——实现界面 UI。

14.1.1　界面规划

由软件功能我们大略地可以做出如下设计：

（1）主界面：进入主界面中，用户可以选择新建跟踪或查看已有跟踪，当然也可以退出程序。

（2）新建跟踪界面：我们需要两个编辑框以方便用户输入相关信息，如新建跟踪名、新建跟踪的描述。当然，我们还需要添加一个“确定”按钮，单击该按钮后用户进入地图界面。

（3）已有跟踪界面：如果用户查看已有跟踪，我们需要一个列表，列出数据库中所有的跟踪记录，尽可能详尽地列出所有相关信息，单击其中的任意一条记录同样进入地图界面。

（4）地图界面：首先要显示 Google 地图，接着还需分为两种状况：一是如果是从查看已有跟踪界面跳转过来的，则在地图上显示曾经经历的轨迹；二是如果是从新建跟踪界面跳转过来的，则不做其他操作，等待消息更新界面。

综上所述，程序总共需要 4 个 Activity。关于主界面，这里就偷懒使用了第 13 章中的登录界面，只需修改一下背景图片，这里就不再给出详细代码了，最后显示效果如图 14.1 所示。



图 14.1　主界面

14.1.2　实现新建跟踪界面

新建跟踪时，我们在界面中需要与用户交互的组件主要有 3 个，分别是两个编辑框和一个按钮。其中一个编辑框用来给用户输入跟踪名；另一个编辑框用来输入跟踪描述，如“从 A 点到 B 点”。按钮则是用来确定用户是否编辑完毕，并跳转到地图跟踪界面。该界面比较简单，只需在一个 LinearLayout 中添加组件就可以了，主要代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical"
   >
   <TextView
   android:layout width="fill parent"
   android:layout height="wrap content"
   android:text="新建跟踪："
   android:textSize="30sp"
   />

   <TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="跟踪名："
   android:textSize="25sp"
   />
   <EditText
   android:id="@+id/et1"
   android:layout width="fill parent"
   android:layout height="wrap content"
   android:text="我的跟踪"
   />

   <TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="描述："
   android:textSize="25sp"
   />
   <EditText
   android:id="@+id/et2"
   android:layout width="fill parent"
   android:layout_height="wrap_content"
   android:text="从 A 点到 B 点"
   />
   <Button
   android:id="@+id/btn"
   android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content"
    android:text="确定"
    />
</LinearLayout>
```

最后的显示效果如图 14.2 所示，当然如果你觉得黑色的背景太单调了，那么也可以自己添加一些背景。

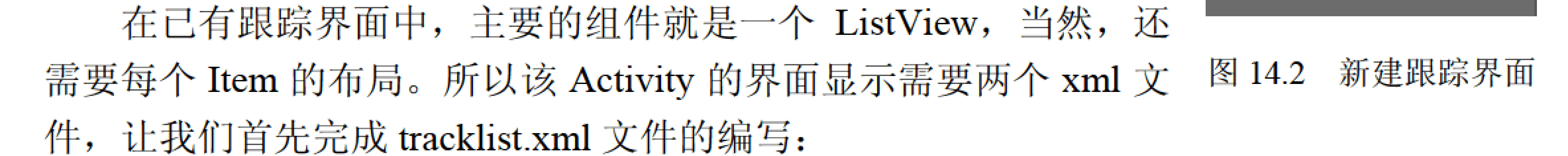


图 14.2　新建跟踪界面

14.1.3　实现已有跟踪界面

在已有跟踪界面中，主要的组件就是一个 ListView，当然，还需要每个 Item 的布局。所以该 Activity 的界面显示需要两个 xml 文件，让我们首先完成 tracklist.xml 文件的编写：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    >
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="已有跟踪："
    android:textSize="25sp"
    />

    <ListView
    android:id="@+id/lv"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

接着，我们完成每个 Item 的布局，每个选项我们希望显示的信息包括：

（1）跟踪名，这是最主要的信息。

（2）跟踪描述，用来显示该记录的相关描述。

（3）开始时间，当本次跟踪的 GPS 开始工作时会记录开始时间。

（4）结束时间，当用户退出软件时默认结束跟踪，记录结束时间。

（5）距离，显示本次跟踪一共记录行径的里程数。

（6）速度，用距离除以持续的时间，我们可以得到一个平均值，该值就是平均速度。

（7）记录点数：显示本次记录一共记录了多少个 GeoPoint 点。

所以 list_item.xml 的代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    >
    <TextView
    android:id="@+id/tv1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
     android:paddingLeft="10dp"
```

```
    />

    <TextView
    android:id="@+id/tv2"
    android:layout width="fill parent"
    android:layout height="wrap content"
     android:paddingLeft="10dp"
    />

   ......        //省略了其他 5 个 TextView 的显示，它们唯一不同的属性就是 id
</LinearLayout>
```

现在还看不出界面的布局效果，不过为了本小节内容的完整，也使大家更直观，这里先给出运行效果，如图 14.3 所示。



图 14.3　已有跟踪界面

14.1.4　实现地图显示界面

地图界面中除了最主要的组件 MapView 之外，我们还需要一些其他的组件来丰富地图功能：

（1）地址输入框，用来输入地址。

（2）“查询”按钮，单击后开始查询该地址，并将地图中心设为该点。

（3）“我的位置”按钮，单击后地图返回到我的位置。

（4）“放大”按钮，单击后放大地图。

（5）“缩小”按钮，单击后缩小地图。

最终的 track.xml 文件代码如下：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
    <com.google.android.maps.MapView
    android:id="@+id/mv"
    android:clickable="true"
    android:layout width="fill parent"
    android:layout height="fill parent"
    android:apiKey="0nEgBgK-FjWmoXZap7PhTSm2PnjP-Gw1jTzUogQ"
    />
    <LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_gravity="top">
    <EditText
    android:id="@+id/et"
    android:layout width="wrap content"
    android:layout_height="wrap_content"
    android:text="苏州虎丘"
    />
    <Button
    android:id="@+id/search"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="search"
    />
    </LinearLayout>
    <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_gravity="bottom|right">
    <Button
    android:id="@+id/myaddr"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:text="我的位置"
    android:textSize="8sp"
    android:paddingRight="5dp"
    />
    <Button
    android:id="@+id/zoomin"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:text="close"
    android:paddingRight="5dp"
    />
    <Button
    android:id="@+id/zoomout"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:text="far"
    android:paddingLeft="5dp"
    />
    </LinearLayout>
</FrameLayout>
```

这里还需提醒读者的是：MapView 的 apiKey 属性必须填写；MapView 的 clickable 属性 N 必须为 true，否则地图无法拖曳，最后效果显示如图 14.4 所示。

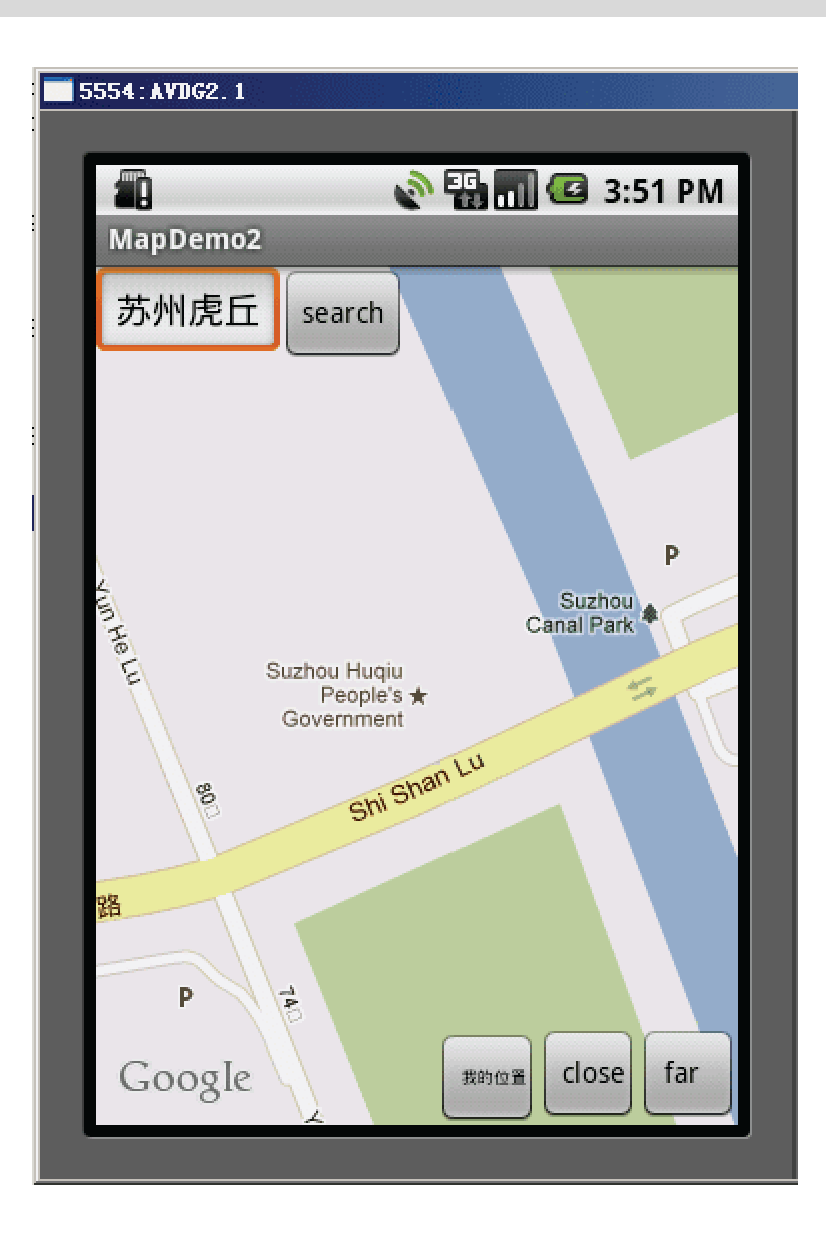


图 14.4　显示地图

14.2　数据库实现

按照功能需求，我们需要在数据库中设计两张表，一张表用来记录所有的 GeoPoint 点的相关信息，我们将它命名为 geopoints；另一张表用来记录每条记录的信息，命名为 tracks。通过这样的设计，当用户要查看已有跟踪时只需查询 tracks 表，而地图显示要绘制路径时，只需查询 geopoints 表。

14.2.1　设计表结构

表 geopoints 中需要列出所有的点，包含的信息有 ID、经度、纬度、创建时间、所属跟踪名等。所以我们将表设计如表 14-1 所示。

表 14-1　geopoints表结构

属　　性	类　　型	含　　义
Id	INTEGER	主键
Latitude	Text	经度
Longitude	Text	纬度
CreateTime	Text	创建时间
trackName	Text	所属的跟踪名

表 tracks 中需要列出所有的跟踪记录，需要包含的信息有 ID、跟踪名、跟踪描述、开始时间、结束时间、距离、速度、跟踪点数等，所以表结构如表 14-2 所示。

表 14-2　tracks表结构

属　　性	类　　型	含　　义
ID	INTEGER	主键
Name	Text	本次跟踪的名字
Description	Text	本次跟踪的附加信息
Createtime	Text	创建时间
Endtime	Text	结束时间
Distance	Text	本次记录的距离
Speed	Text	平均速度
Count	Text	记录的点数

由于 SQLite 数据库并没有对属性的类型进行严格的规定，所以在设计简单数据库表的时候，我们将之都设定为了 Text，这样以后在使用时也可以方便地转换类型。

14.2.2　实现 DatabaseHelper

根据以上两张表的分析，我们可以实现 DatabaseHelper 的编写。关于 SQLiteOpenHelper 不知道读者是否已经掌握。如果还有疑惑，请翻阅本书的第 9 章——Android 中的数据存储。最后的 DatabaseHelper 的代码如下所示：

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHelper  extends SQLiteOpenHelper
{
    final static String DATABASENAME       = "my_database.db";
    final static int    VERSION            = 1;
    final static String TABLENAME          = "geopoints";
    final static String GEO_ID             = "id";
    final static String GEO_LATITUDE       = "latitude";
    final static String GEO_LONGITUDE      = "longitude";
    final static String GEO_TIME           = "time";
    final static String GEO_TRACKNAME      = "trackname";
    final static String TABLENAME_2        = "tracks";
    final static String TRACK_ID           = "id";
    final static String TRACK_CREATETIME   = "createtime";
    final static String TRACK_NAME         = "name";
```

```
    final static String TRACK_DESC          = "description";
    final static String TRACK_DIST          = "distance";
    final static String TRACK_SPEED         = "speed";
    final static String TRACK_COUNT         = "count";
    final static String TRACK_ENDTIME       = "endtime";
    public DatabaseHelper(Context context)
    {
        super(context, DATABASENAME, null, VERSION);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void onCreate(SQLiteDatabase db)
    {
//实现 geopoints 表
        String sql = "CREATE TABLE " +
        TABLENAME    + "(" +
        GEO_ID          + " INTEGER PRIMARY KEY   AUTOINCREMENT," +
        GEO_LATITUDE    + " TEXT," +
        GEO_LONGITUDE   + " TEXT," +
        GEO_TRACKNAME   + " TEXT," +
        GEO_TIME        + " TEXT);";
        db.execSQL(sql);
        //实现 tracks 表
        String sql2 = "CREATE TABLE " +
        TABLENAME_2          + "(" +
        TRACK_ID             + " INTEGER PRIMARY KEY AUTOINCREMENT," +
        TRACK_NAME           + " TEXT," +
        TRACK_CREATETIME     + " TEXT," +
        TRACK_DESC           + " TEXT," +
        TRACK_DIST           + " TEXT," +
        TRACK_SPEED          + " TEXT," +
        TRACK_COUNT          + " TEXT," +
        TRACK_ENDTIME        + " TEXT);";
        db.execSQL(sql2);
    }

    @Override
    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2)
    {

    }
}
```

14.3　功 能 实 现

设计好数据库以后，我们就可以实现本应用的功能了。本例使用了服务来完成核心工作，我们将之命名为 TrackService。它一旦开始运行后就在后台不停地读取位置信息，读取到的位置信息会写入到数据库中，并通知 Activity 进行相关的界面修改。在地图上也就是 TrackerActivity 中画出所有经历过的点，并用线将它们连接起来，这条线就是我们的轨迹了。

14.3.1　实现 TrackService

本例中的 Service 是需要一直运行在后台的，并不需要与 Activity 绑定，所以在实现时，我们只需重写其中的 onCreate()、onStart()以及 onDestroy()3 个方法。

在 onCreate()方法中，其整体结构如下所示：

1. 整体结构

```
import ......                           //省略部分导入
import android.app.Service;
import android.database.sqlite.SQLiteDatabase;
import android.location.Criteria;
import android.location.LocationListener;
import android.location.LocationManager;

public class TrackService extends Service
{
    public static final int MSG = 1;
    private DatabaseHelper helper;
    private SQLiteDatabase db;
    private LocationManager manager;
    private String locationProvider;
    private LocationListener locationListener;
    private long distance = 0;
    private long speed = 0;
    private long endTime;
    private long createTime = 0;
private List<Location> list = new ArrayList<Location>();

    @Override
    public IBinder onBind(Intent intent)
    {
        return null;
    }

    @Override
    public void onCreate()
    {
createTime = System.currentTimeMillis();              //得到开始时间
//此处完成 SQLiteDatabas、LocationListener、LocationManager、LocationProvider
的初始化工作
        super.onCreate();
    }

    @Override
    public void onStart(Intent intent, int startId)
    {
         //开始监听位置变化信息
       manager.requestLocationUpdates(locationProvider, 1000, 10,
       locationListener);
        super.onStart(intent, startId);
    }

    @Override
    public void onDestroy()
    {
```

```
        endTime = System.currentTimeMillis();       //得到结束时间
        insertTrackInfo();                          //添加到数据库中
        super.onDestroy();
    }
}
```

2. 完成onCreate()方法

接下来我们就完成 onCreate()方法中需要完成的各类初始化工作。其代码如下：

```
@Override
    public void onCreate()
    {
        helper = new DatabaseHelper(getBaseContext());
        db = helper.getWritableDatabase();
        //得到 LocationManager 对象
       manager = (LocationManager) getSystemService(Context.LOCATION
       SERVICE);
       //得到提供商
       locationProvider = getLocationProvider(manager);
       //新建位置监听器
       locationListener = new LocationListener()
       {
        @Override
        public void onLocationChanged(Location location)
        {
            Message msg = TrackerActivity.trackHandler.obtainMessage();
            msg.what = MSG;
            msg.arg1 = list.size();
            msg.obj = location;
            TrackerActivity.trackHandler.sendMessage(msg);
            insertGeoPoint(location);           //保存到数据库
        }

        @Override
        public void onStatusChanged(String provider, int status, Bundle
        extras)
        {
        }

        @Override
        public void onProviderEnabled(String provider)
        {
        }

        @Override
        public void onProviderDisabled(String provider)
        {
        }
       };

        super.onCreate();
    }
```

3. 完成getLocationProvider()方法

getLocationProvider()方法如下。该方法在第 12 章中已经有过讲解，如果有疑问请参照第 12 章一起阅读。

```
public String getLocationProvider(LocationManager lm)
{
    //新建一个选择提供商的标准
    Criteria cri = new Criteria();
    //设置精确度为精确，还可以设置为粗略 ACCURACY.COARSE
    cri.setAccuracy(Criteria.ACCURACY_FINE);
    //设置耗电等级为低
    cri.setPowerRequirement(Criteria.POWER_LOW);
    //设置是否向用户收费
    cri.setCostAllowed(true);
    //设置是否需要海拔信息
    cri.setAltitudeRequired(false);
    //设置是否需要方位信息
    cri.setBearingRequired(false);
    //得到最好的内容提供商
    String lp = lm.getBestProvider(cri, true);
    return lp;
}
```

4．完成insertGeoPoint()方法

在 locationListener 中，每当得到一个新的 Location 时，我们需要执行两个操作：

（1）将 location 信息交给 TrackerActivity。

（2）将 location 信息写入数据库。

实现第一步我们使用了 Handler 机制，而实现第二步则是调用了 insertGeoPoint()方法，该方法代码如下所示：

```
public long insertGeoPoint(Location location)
{
    ContentValues values = new ContentValues();
    values.put(DatabaseHelper.GEO_LATITUDE, location.getLatitude());
    values.put(DatabaseHelper.GEO_LONGITUDE, location.getLongitude());
    values.put(DatabaseHelper.GEO_TIME, System.currentTimeMillis());
    values.put(DatabaseHelper.GEO_TRACKNAME, NewTrackActivity.name);
    long rowId = db.insert(DatabaseHelper.TABLENAME, null, values);
    return rowId;
}
```

5．完成insertTrackInfo()方法

在 Service 结束时，也就是 onDestroy()方法结束时，我们需要将本次记录的相关信息保存到 tracks 表中，方法如下：

```
public long insertTrackInfo()
{
    ContentValues values = new ContentValues();
    values.put(DatabaseHelper.TRACK_NAME, NewTrackActivity.name);
    values.put(DatabaseHelper.TRACK_DESC, NewTrackActivity.desc);
    values.put(DatabaseHelper.TRACK_CREATETIME, createTime);
    values.put(DatabaseHelper.TRACK_ENDTIME, endTime);
    values.put(DatabaseHelper.TRACK_COUNT, list.size());
    values.put(DatabaseHelper.TRACK_DIST, getDistance());
    values.put(DatabaseHelper.TRACK_SPEED, getSpeed());
    long rowId = db.insertOrThrow(DatabaseHelper.TABLENAME_2, null, values);
    Log.i("TAG",String.valueOf(rowId));
    return rowId;
}
```

其中还有两个小方法，分别是 getSpeed()和 getDistance()。其中比较关键的是getDistance()方法，因为 getSpeed()只需将距离除以持续时间就可以了。如下所示：

```
public long getSpeed()
{
    long during = endTime - createTime;                //单位：s
    speed = distance/during;
    return speed;
}
```

那么 getDistance()又是如何实现的呢？其算法是：简单地将距离计算成为了第一个点和最后一个点之间的距离，这样计算出来的结果误差也许会比较大，如果读者有兴趣的话，可以优化该算法，计算每两个点之间的距离，而后将之相加。

```
public double getDistance()
{
    GeoPoint p1 = list.get(0);
    GeoPoint p2 = list.get(list.size() - 1);
    double lat1 = (Math.PI/180)*(p1.getLatitudeE6()/1E6);
    double lon1 = (Math.PI/180)*(p1.getLongitudeE6()/1E6);
    double lat2 = (Math.PI/180)*(p2.getLatitudeE6()/1E6);
    double lon2 = (Math.PI/180)*(p2.getLongitudeE6()/1E6);
    double R = 6371;                  //地球半径，单位：KM
    distance = Math.acos(Math.sin(lat1)*Math.sin(lat2)+Math.cos
    (lat1)*Math.cos(lat2)*Math.cos(lon2-lon1))*R*1000;
    return distance;                  //单位：m
}
```

14.3.2　实现 OldTrackActivity

OldTrackActivity 的功能主要有两个：

（1）显示已有跟踪。

（2）单击任意跟踪，进入地图界面。

在 14.3.1 小节中实现的 Service 中，每次结束时都会将记录添加到 tracks 表中，所以每次用户登录时都可以查看已有跟踪。其核心方法就是执行数据库查询操作。

1．整体设计

我们首先完成其整体的结构设计，整体设计中最主要的工作就是界面的初始化：

```
Import ......               //省略部分导入
import java.text.DateFormat;
import java.util.Date;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListAdapter;
import android.widget.ListView;

public class OldTrackActivity extends Activity
{
    public static String name = null;
    public static String desc = null;
    public static int   createTime = 0;
```

```
    ListView lv;
    SQLiteDatabase db;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.tracklist);
        //初始化数据库
        DatabaseHelper helper = new DatabaseHelper(getBaseContext());
        db = helper.getReadableDatabase();
        //初始化界面
        initView();
    }
}
```

2．完成initView()方法

InitView()方法中实现了所有的界面初始化和单击事件的监听。其中的 getData()方法得到了要显示的数据源，通过该数据源可以产生一个适配器，使用该适配器就可以将数据与界面很好地绑定了。

```
public void initView()
    {
        lv = (ListView) findViewById(R.id.lv);
        final ArrayList<HashMap<String, String>> data = getData();
                                               //得到数据源
        ListAdapter adapter = new SimpleAdapter(getBaseContext(),
                                               //得到适配器
                data,
                R.layout.list_item,
                new String[]{"trackName","trackDesc","trackBegTime",
                "trackEndTime","trackDist","trackSpeed","trackCount"},
                new int[]{R.id.tv1,R.id.tv2,R.id.tv3,R.id.tv4,R.id.tv5,
                R.id.tv6,R.id.tv7,});
        lv.setAdapter( adapter);                //设置适配器
        //监听单击事件
        lv.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int
            position, long id)
            {
                String trackName = data.get(position).get("trackName");
                Intent i = new Intent(OldTrackActivity.
                this,TrackerActivity.class);
                i.putExtra("trackName", trackName);
                startActivity(i);
            }
        });
    }
```

3．完成getData()方法

该方法可以说是 Activity 的核心了，其工作是查询数据库得到返回的 Cursor 对象，然后逐条遍历将之重新构造为我们需要的数据结构。

```
public ArrayList<HashMap<String, String>> getData()
    {
```

```
        ArrayList<HashMap<String, String>> listItems= new ArrayList<
        HashMap<String, String>>();
        Cursor cursor = db.query(DatabaseHelper.TABLENAME 2,
                null,null,null,null, null, null);
        //cursor 读取第一条记录
        cursor.moveToFirst();
        while(!cursor.isAfterLast())
        {
            //读取所有存储的内容
            String trackId   = cursor.getString(cursor.getColumnIndex
            (DatabaseHelper.TRACK ID));
            String trackName  = cursor.getString(cursor.getColumnIndex
            (DatabaseHelper.TRACK NAME));
            String trackDesc  = cursor.getString(cursor.getColumnIndex
            (DatabaseHelper.TRACK DESC));
            long trackBegTime  = cursor.getLong(cursor.getColumnIndex
            (DatabaseHelper.TRACK CREATETIME));
            long trackEndTime  = cursor.getLong(cursor.getColumnIndex
            (DatabaseHelper.TRACK ENDTIME));
            String trackDist  = cursor.getString(cursor.getColumnIndex
            (DatabaseHelper.TRACK_DIST));
            String trackSpeed = cursor.getString(cursor.getColumnIndex
            (DatabaseHelper.TRACK_SPEED));
            String trackCount = cursor.getString(cursor.getColumnIndex
            (DatabaseHelper.TRACK_COUNT));
            //将内容重新组织
            Date begDate = new Date(trackBegTime);
            String trackBegDate = DateFormat.getDateTimeInstance().
            format(begDate);
            Date endDate = new Date(trackEndTime);
            String trackEndDate = DateFormat.getDateTimeInstance().
            format(endDate);
            HashMap<String, String> map = new HashMap<String, String>();
            map.put("trackId",trackId);
            map.put("trackName","跟踪名: "+trackName);
            map.put("trackDesc","描述: "+trackDesc);
            map.put("trackBegTime","开始时间: "+trackBegDate);
            map.put("trackEndTime","结束时间: "+trackEndDate);
            map.put("trackDist","距离: "+trackDist + " km");
            map.put("trackSpeed","速度: "+trackSpeed + " km/h");
            map.put("trackCount","记录点数: "+trackCount);
            //添加到列表中
            listItems.add(map);
            //读取下一个
            cursor.moveToNext();
        }
        return listItems;
    }
```

14.3.3　实现 TrackerActivity

在 TrackerActivity 中我们首先完成地图的基本功能，如放大、缩小、切换视角等。

1．整体结构

```
import ......                             //省略部分导入
import com.google.android.maps.GeoPoint;
```

```
import com.google.android.maps.MapActivity;
import com.google.android.maps.Overlay;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;

public class TrackerActivity extends MapActivity {
   MapView mv;
   MapController controller;
   GeoPoint curPoint;
   Button b_zoomin;
   Button b_zoomout;
   Button b_myaddr;
   Button b_search;
   EditText et;
   TextView pop_content;
   View popView;
   public static Handler  trackHandler;
   public static final int OLD_TRACK   = 0;
   public static final int STREET_ID   = 1;
   public static final int TRAFFIC_ID = 2;
   public static final int SATE_ID     = 3;
   public static final int DEFAULT_ID = 4;


   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.track);
      //初始化界面
      initView();
      //新建 handler
      trackHandler = new Handler()
      {
           @Override
           public void handleMessage(Message msg)
           {
               //这里根据不同的消息进行界面的相关修改
}
      };
      //判断是否从已有记录界面跳转过来
      String trackName = getIntent().getStringExtra("trackName");
      if (trackName != null)
      {
       Message msg = trackHandler.obtainMessage();
       msg.what = OLD_TRACK;
       msg.obj  = trackName;
       trackHandler.sendMessage(msg);
      }
else
{
//开始服务，在后台记录地理位置的变更
        Intent i = new Intent(this,TrackService.class);
  startService(i);
}

   }
```

```
    @Override
    protected boolean isRouteDisplayed()
    {
        // TODO Auto-generated method stub
        return false;
    }
    //添加菜单项，增加各个视图
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);

        menu.add(0, TRAFFIC_ID, 0, "交通");
        menu.add(0, SATE_ID,    1, "卫星");
        menu.add(0, STREET_ID,  2, "街道");
        menu.add(0, DEFAULT_ID, 3, "默认");
        return true;
    }
//增加视图的实现
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case TRAFFIC_ID:
            {
              mv.setTraffic(true);
              mv.setSatellite(false);
              mv.setStreetView(false);
              return true;
            }
            case SATE_ID:
            {
             mv.setSatellite(true);
             mv.setStreetView(false);
             mv.setTraffic(false);
               return true;
            }
            case STREET_ID:
            {
              mv.setStreetView(true);
              mv.setSatellite(false);
              mv.setTraffic(false);
              return true;
            }
            case DEFAULT_ID:
            {
              mv.setStreetView(false);
              mv.setSatellite(false);
              mv.setTraffic(false);
              return true;
            }

        }
        return super.onOptionsItemSelected(item);
    }
```

以上代码的加粗部分用来判断本界面的上一个界面是否是 OldTrackActivity，如果是则可以从 Intent 中得到附带的 trackName，如果不是则得到的值为空。

如果是，则向 Handler 发送一条消息，通知 Handler 进行界面的修改，也就是查询该 trackname 的所有 geopoint，将它们画在地图上并用线条连接形成轨迹。

2．实现Handler

Handler 需要执行两种操作：

（1）如果消息来自 Service，也就是其类型为 TrackService.MSG，则在地图上画出该点。

（2）如果消息的类型是 OLD_TRACK，则在地图上将其轨迹画出。

```
@Override
            public void handleMessage(Message msg)
            {
                if (msg.what == TrackService.MSG)
                {
                    int count = msg.arg1;
                    String pointName = "位置"+String.valueOf(count);
                    Location location = (Location) msg.obj;
                    //从 Location 中获得 GeoPoint 对象
                    curPoint = getGeoPoint(location);
                    //将地图移动到该地点
                    controller.animateTo(curPoint);
                    //删除之前的 pop
                    mv.removeView(popView);
                    //设置 popView 的属性
                    popView.setLayoutParams(new MapView.LayoutParams
                    (MapView.LayoutParams.WRAP_CONTENT, MapView.Layout-
                        Params.WRAP_CONTENT, curPoint, MapView.Layout-
                        Params.BOTTOM_CENTER));
                    //在气泡中要显示的内容
                    pop_content.setText("我的位置");
                    //将 popView 添加到地图中
                    mv.addView(popView);
                    //使用 Overlay 显示
                    MyOverlay overlay = new MyOverlay(curPoint,pointName);
                    //得到已有的 Overlay 列表
                    List<Overlay> overlays = mv.getOverlays();
                    //将新建的 Overlay 加入到列表中显示
                    overlays.add(overlay);
                }
                else if (msg.what == OLD_TRACK)
                {
                    //得到已有的 Overlay 列表
                    List<Overlay> overlays = mv.getOverlays();
                    String trackName = (String) msg.obj;
                    DatabaseHelper helper = new DatabaseHelper
                    (getBaseContext());
                    SQLiteDatabase db = helper.getWritableDatabase();
                    Cursor  cursor  =  db.query(DatabaseHelper.TABLENAME,
null, DatabaseHelper.GEO_TRACKNAME + "=?", new String[]{trackName}, null,
null, null);
                    cursor.moveToFirst();
                    while (!cursor.isAfterLast())
                    {
                        int latitude  = (int)  (Double.parseDouble(cursor.
                        etString(cursor.getColumnIndex(DatabaseHelper.
                        GEO_LATITUDE)))*1E6);
                        int longitude = (int)  (Double.parseDouble(cursor.
                        getString(cursor.getColumnIndex(DatabaseHelper.
                        GEO_LONGITUDE)))*1E6);
                        String geoId = cursor.getString(cursor.getColumnIndex
```

```
                        (DatabaseHelper.GEO ID));
                        Log.i("TAG","id:"+geoId);
                        Log.i("TAG",String.valueOf(latitude)+":"+String.
                        valueOf(longitude));
                        GeoPoint point = new GeoPoint(latitude,longitude);
                        controller.animateTo(point);
                        //使用 Overlay 显示
                        MyOverlay overlay = new MyOverlay(point,"位置
                        "+geoId);
                        //将新建的 Overlay 加入到列表中显示
                        overlays.add(overlay);
                        cursor.moveToNext();
                    }
                    cursor.close();
                    db.close();
                }
                super.handleMessage(msg);
            }
```

3. 实现initView()方法

initView()方法实现了组件的初始化以及各个按钮的单击事件。这里仅仅给出了各个单击事件的相关代码，省略了各个组件的 findViewById()方法，相信大家可以自行完成。如有问题请参照源代码。

```
public void initView()
{
    //实例化 pop 布局
    LayoutInflater inflater = LayoutInflater.from(this);
    popView = inflater.inflate(R.layout.pop, null);
    popView.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            v.setVisibility(View.GONE);
        }
    });
    ......                                  //此处省略了各个组件实例化
    //得到 MapView 的控制器
   controller = mv.getController();
   //设置地图缩放等级，参数的值在 1～21 之间
   controller.setZoom(10);
    OnClickListener l = new OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            int id = v.getId();
            switch(id)
            {
            case R.id.zoomin:
            {
                zoomIn();
                break;
            }
            case R.id.zoomout:
            {
                zoomOut();
```

```
                break;
            }
            case R.id.myaddr:
            {
                if (curPoint != null)
                {
                    controller.animateTo(curPoint);
                }
                else
                {
                    Toast.makeText(getBaseContext(), "暂时还未获得您的位置",
                    Toast.LENGTH SHORT).show();
                }
                break;
            }
            case R.id.search:
            {
                String addrName =  et.getText().toString();
                GeoPoint point = getGeoPointByAddr(addrName);
                controller.animateTo(point);
                controller.setZoom(20);
                break;
            }
            }
        }
    };
    //设置各个按钮的单击事件
    b zoomin.setOnClickListener(l);
    b zoomout.setOnClickListener(l);
    b myaddr.setOnClickListener(l);
    b search.setOnClickListener(l);
}
```

4．实现地理位置与经纬度坐标的转换

一般所有的地图都有查询功能，该功能需要将经纬度转换为实际的地名，这样显示更直观；同时也需要将实际地名转换为经纬度坐标点，这样地图才知道怎样显示。为了以后使用方便，这里写了两个方法：

（1）getAddrByGeoPoint()，用来将 GeoPoint 对象转换为实际地址。

（2）getGeoPointByAddr()，用来将实际地址反转为 GeoPoint 对象。

这两个方法的实现代码如下：

```
    public String getAddrByGeoPoint(GeoPoint point)
    {
        String addr = "";
        try
        {
            //新建解码器
            Geocoder coder = new Geocoder(this,Locale.getDefault());
            //得到经纬度
            double latitude = point.getLatitudeE6()/1E6;
            double longitude = point.getLongitudeE6()/1E6;
            //得到地址
            List<Address> list = coder.getFromLocation(latitude, longitude, 1);
            StringBuilder builder = new StringBuilder();
            if (list.size() > 0)
            {
```

```
                Address address = list.get(0);
                for(int i = 0; i < address.getMaxAddressLineIndex();i++)
                {
                    builder.append(address.getAddressLine(i)+"\n");
                }
                //得到国家
                builder.append("国家: "+address.getCountryName());
                addr = builder.toString();
                Log.i("TAG", addr);
            }
            else
            {
                Toast.makeText(getBaseContext(), "无法解析到地名", Toast.
                LENGTH_SHORT).show();
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        return addr;
    }

    public GeoPoint getGeoPointByAddr(String addr)
    {
        GeoPoint point = null;
        try
        {
            //新建解码器
            Geocoder coder = new Geocoder(this,Locale.getDefault());
            //得到地址
            List<Address> list = coder.getFromLocationName(addr, 1);
            if (list.size() > 0)
            {
                Address address = list.get(0);
                //得到经纬度
                double latitude = address.getLatitude()*1E6;
                double longitude = address.getLongitude()*1E6;
                //得到 GeoPoint 对象
                point = new GeoPoint((int)latitude,(int)longitude);
            }
            else
            {
                Toast.makeText(getBaseContext(), "无法解析地名", Toast.
                LENGTH_SHORT).show();
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        return point;
    }
}
```

14.3.4　实现 Overlay

Overlay 是用来在地图上画各类图形的，为了能画出轨迹，我们需要在 draw()方法中实

现 3 个工作：

（1）画出一个小红点，用来标注用户曾经经过的地点，主要方法为：

```
canvas.drawOval(rect,paint);
```

（2）在小红点边上画出说明性文字和背景色，主要方法为：

```
canvas.drawRoundRect(backRect, 5, 5, backPaint);              //画背景
canvas.drawText(content, point.x+10, point.y, textPaint);     //写文字
```

（3）将这些小红点连接起来，形成个人的轨迹，主要方法为：

```
canvas.drawPath(path, paint);
```

最后，MyOverlay 的实现代码如下：

```
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.RectF;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.Projection;

public class MyOverlay extends Overlay
{
    long latitude;
    long longitude;
    List<Point> list = new ArrayList<Point>();
    String content;

    public MyOverlay(long latitude,long longitude,String content)
    {
        this.latitude = latitude;
        this.longitude  =  longitude;
        this.content = content;
    }

    public MyOverlay(GeoPoint point,String content)
    {
        this.latitude   = point.getLatitudeE6();
        this.longitude  =  point.getLongitudeE6();
        this.content = content;
    }

    @Override
    public boolean draw(Canvas canvas, MapView mapview, boolean shadow, long when)
    {
        //新建投影数据对象
        Projection projection = mapview.getProjection();
        //新建 GeoPoint 对象
        GeoPoint geoPoint = new GeoPoint((int)latitude,(int)longitude);
        //新建 Point 对象
        Point point = new Point();
        //将 GeoPoint 对象投影成可显示的 Point 对象
        projection.toPixels(geoPoint,point);
```

```
        list.add(point);
        //创建一个 RectF 对象
        RectF rect = new RectF(point.x-5,point.y-5,point.x +5,point.y+5);
        //新建颜料对象
        Paint paint = new Paint();
        //设置颜色为红色
        paint.setColor(Color.RED);
        //在画布上将椭圆画出来
        canvas.drawOval(rect,paint);

        //创建背景
        RectF backRect = new RectF(point.x+7,point.y-15,point.x +60,
        point.y+5);
        //新建背景的颜料对象
        Paint backPaint = new Paint();
        //设置字体颜色为白色
        backPaint.setColor(Color.GRAY);
        //设置透明度
        backPaint.setAlpha(100);
        //在画布上画出一个圆角矩形
        canvas.drawRoundRect(backRect, 5, 5, backPaint);

        //新建字体的颜料对象
        Paint textPaint = new Paint();
        //设置字体颜色为白色
        backPaint.setColor(Color.WHITE);
        //将内容显示在画布上，坐标点要在背景之间
        canvas.drawText(content, point.x+10, point.y, textPaint);
        //画出轨迹
        for (int i = 0;i< list.size() - 1;i++)
        {
            //得到前一个点
            GeoPoint oldGeoPoint = list.get(i);
            Point oldPoint = new Point();
            projection.toPixels(oldGeoPoint,oldPoint);
            //得到后一个点
            GeoPoint newGeoPoint = list.get(i+1);
            Point newPoint = new Point();
            projection.toPixels(newGeoPoint,newPoint);
            //将轨迹画出
            Path path = new Path();
            path.moveTo(oldPoint.x, oldPoint.y);
            path.lineTo(newPoint.x, newPoint.y);
            canvas.drawPath(path, paint);
        }
        return super.draw(canvas, mapview, shadow, when);
    }
}
```

14.3.5　修改注册文件

到这里，主要的代码已经完成了，最后不要忘记在注册文件中添加相关权限、Activity 信息以及 Service 信息：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.wes.tracker"
     android:versionCode="1"
     android:versionName="1.0">
   <uses-sdk android:minSdkVersion="7" />

   <application android:icon="@drawable/icon" android:label=
   "@string/app name">
      <activity android:name="com.wes.tracker.MainActivity"
              android:label="@string/app name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
       <activity android:name="com.wes.tracker.NewTrackActivity"/>
       <activity android:name="com.wes.tracker.OldTrackActivity"/>
       <activity android:name="TrackerActivity"/>
       <service android:name="com.wes.tracker.TrackService"/>
       <uses-library android:name="com.google.android.maps"/>
   </application>
   <uses-permission android:name="android.permission.INTERNET"/>
   <uses-permission
android:name="android.permission.ACCESS FINE LOCATION"/>
   <uses-permission
android:name="android.permission.ACCESS COARSE LOCATION"/>
</manifest>
```

现在，程序已经可以运行啦，运行效果如图 14.5 所示。放大后的轨迹如图 14.6 所示。

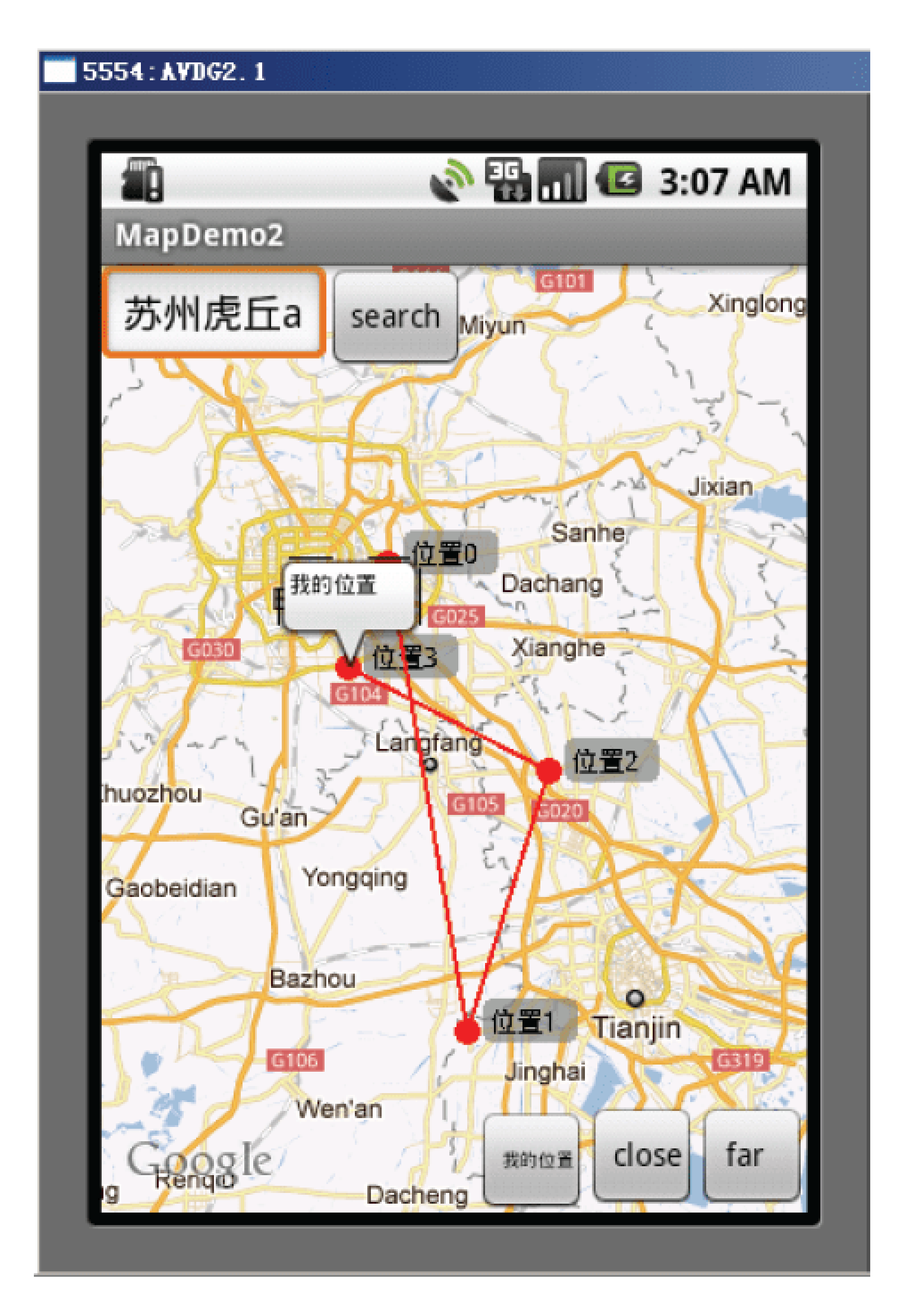


图 14.5　轨迹记录

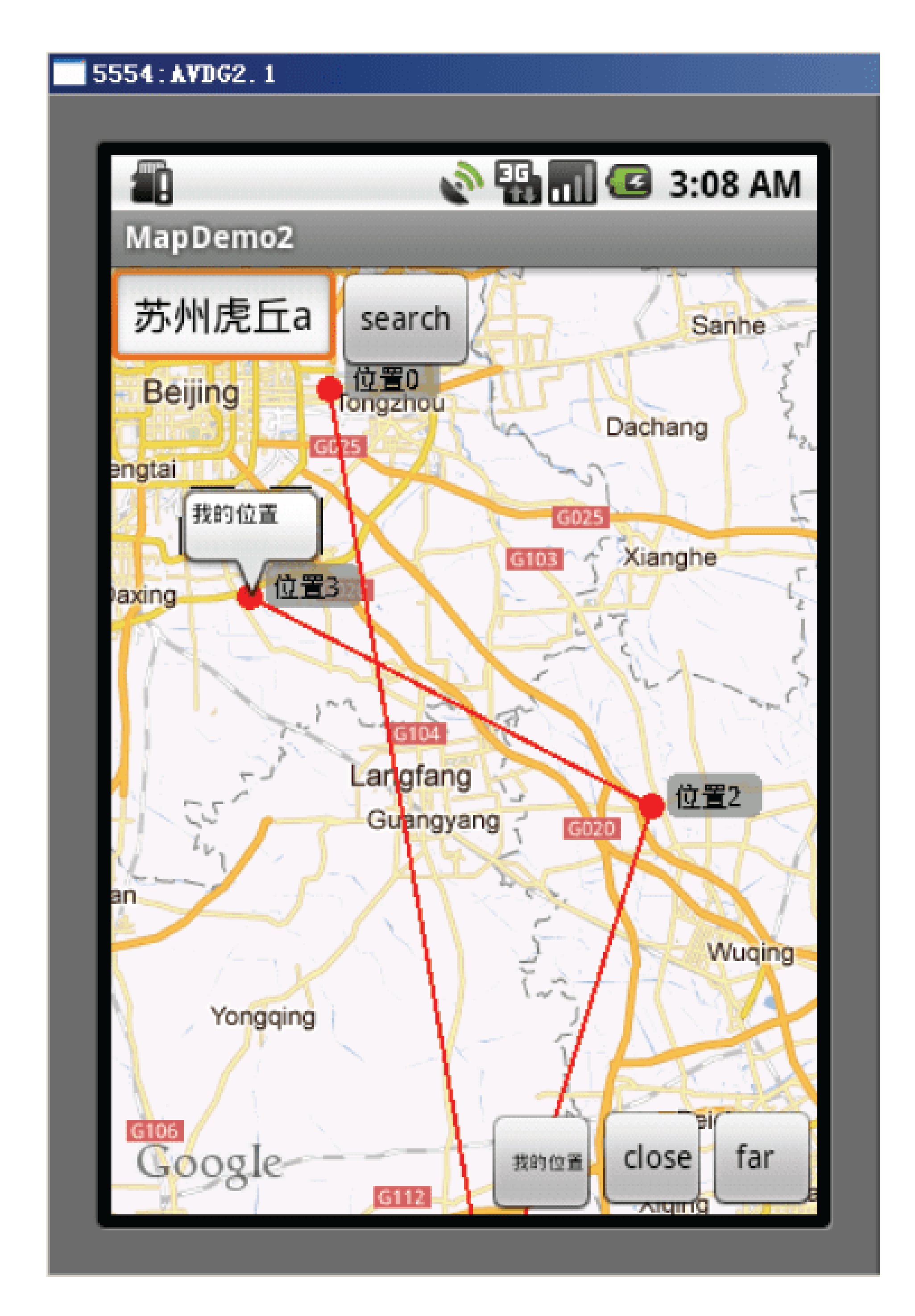


图 14.6　放大后的轨迹

14.4　小　　结

本章我们一起实现了一个 LBS 应用——个人轨迹记录器。通过本章的学习，读者可以巩固数据库、Service、Activity 以及 Google Maps 等相关知识，同时熟悉一个应用从设计到实现的基本流程。本章的重点在于 Google Maps API 的熟练使用，难点是使用 Overlay 在地图上划出各类图形。

当然该应用还有更多的提升空间，如实现距离算法的优化、添加地图设置功能、设置地图缩放等级、地图刷新间隔等等，甚至我们可以将记录导出放到网上与大家一起分享。